

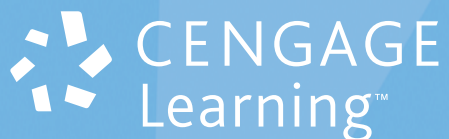


KYLIE DOCHERTY
JUSTIN GRAHAM
ANDREW RUSSELL



NELSON **DIGITAL** **SOLUTIONS**

FOR QCE UNITS 1–4



COPYRIGHT NOTICE

Copyright in this work is owned by Cengage Learning Australia ("the work"). A condition of purchase of this electronic version of the work is that you agree to respect the copyright in the work, abide by the Copyright Act 1968 and specifically agree not to transfer, sell, assign, misuse, copy or transmit an electronic or other version of the work to any third party.

Please note: This product is accompanied by a licence (single user, network or adoption) governing the terms and conditions of its use.

This is a legal agreement between the you, (the “Customer”) and Cengage Learning Australia Pty Limited (ABN 14 058 280 149) (the “Licensor”) which provides the terms and conditions of this non-exclusive licence and the limited warranty for the Product. Use of the Product indicates an acknowledgement that the Customer has read and agreed to be bound by the terms and conditions of this Agreement. If you do not agree to these terms and conditions, return the Product to the place of purchase within 15 days of the date of purchase (with proof of purchase) for a full refund

1. Licence Grant

You do not receive title to the Product. Copyright in the Product (which includes all images, photographs, video, animations, audio, music and text incorporated in the Product, including all of the accompanying printed material) is owned by the Licensor and/or its suppliers and is protected by Australian copyright laws. The Licensor grants you a non-exclusive licence to use the Product subject to the restrictions and terms set out in this Agreement.

2. A Licence allows you to:

Use the Product on your computer. The Customer represents that they shall in no way place the Product in the public domain or in any way compromise our copyright in the Material. You agree to take reasonable steps to protect our copyright.

3. You may not:

Alter, modify, translate, reverse engineer, decompile, or adapt the software or create derivative works based on the Product. Make further copies by any means technological, electronic, digital whatsoever without the written permission of the Licensor. Rent or transfer all or any part of your rights under this Agreement. Remove or alter any copyright or other proprietary notice or label attached to the software.

4. Termination

Any failure to comply with the terms and conditions of this agreement will result in the automatic termination of this licence. Upon termination of this licence for any reason, the Customer must destroy or return to the Licensor all copies of the software and accompanying documentation.

5. Warranties

To the extent permitted by law, the Licensor’s liability for any breach of the warranty or any term implied by law into this licence is limited to the lowest cost of replacing the goods, acquiring equivalent goods or having the goods repaired.

KYLIE DOCHERTY
JUSTIN GRAHAM
ANDREW RUSSELL



NELSON DIGITAL
SOLUTIONS
FOR QCE

Digital Solutions for Queensland Units 1-4

1st Edition

Kylie Docherty

Justin Graham

Andrew Russell

ISBN 9780170420655

Publishers: Lizzie Allmand and Rachel Ford

Project editor: Robyn Beaver

Copy editor: Nick Tapp

Text design: Jennai Lee Fai (Jenki)

Project designer: James Steer

Permissions researcher: Debbie Gallagher

Production controller: Karen Young

Typesetter: Q2A Media

Any URLs contained in this publication were checked for currency during the production process. Note, however, that the publisher cannot vouch for the ongoing currency of URLs.

© 2018 Kylie Docherty, Justin Graham, Andrew Russell

Copyright Notice

This Work is copyright. No part of this Work may be reproduced, stored in a retrieval system, or transmitted in any form or by any means without prior written permission of the Publisher. Except as permitted under the *Copyright Act 1968*, for example any fair dealing for the purposes of private study, research, criticism or review, subject to certain limitations. These limitations include: Restricting the copying to a maximum of one chapter or 10% of this book, whichever is greater; providing an appropriate notice and warning with the copies of the Work disseminated; taking all reasonable steps to limit access to these copies to people authorised to receive these copies; ensuring you hold the appropriate Licences issued by the Copyright Agency Limited ("CAL"), supply a remuneration notice to CAL and pay any required fees. For details of CAL licences and remuneration notices please contact CAL at Level 11, 66 Goulburn Street, Sydney NSW 2000, Tel: (02) 9394 7600, Fax: (02) 9394 7601
Email: info@copyright.com.au
Website: www.copyright.com.au

For product information and technology assistance,
in Australia call **1300 790 853**;
in New Zealand call **0800 449 725**

For permission to use material from this text or product, please email
aust.permissions@cengage.com

National Library of Australia Cataloguing-in-Publication Data

A catalogue record for this book is available from the National Library of Australia.

Cengage Learning Australia

Level 7, 80 Dorcas Street
South Melbourne, Victoria Australia 3205

Cengage Learning New Zealand

Unit 4B Rosedale Office Park
331 Rosedale Road, Albany, North Shore 0632, NZ

For learning solutions, visit cengage.com.au

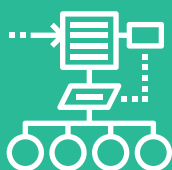
Printed in China by China Translation & Printing Services.

1 2 3 4 5 6 7 22 21 20 19 18



CONTENTS

Introduction	v
About the authors	vi
Implementation advice (for teachers)	vii
The problem-solving process (for students)	x



Unit 1: Creating with code	1
1.1 Exploring digital problems	2
1.2 User experiences and interfaces	22
1.3 Algorithms and programming techniques	35
1.4 Programmed solutions	66



Unit 2: Applications and data solutions	87
2.1 Exploring data-driven applications and solutions	88
2.2 Developing data-driven applications and solutions	122
2.3 Generating data-driven applications and solutions	148
2.4 Evaluating data-driven applications and solutions	181



Unit 3: Digital innovation	193
3.1 Exploring innovation	194
3.2 Developing an innovative solution	217
3.3 Generating a digital prototype	250
3.4 Evaluating a digital solution	277



Unit 4: Digital impacts 283

4.1	Exploring cybersecurity and data exchange	284
4.2	Developing data exchange solutions	310
4.3	Developing and generating methods of data security	326
4.4	Evaluating the impacts of data exchange and open data	349



Glossary	372
Index	375

INTRODUCTION

This first edition of *Digital Solutions for QCE* introduces content for the new senior Digital Solutions subject that will be taught for the first time in Queensland schools in 2019. The textbook steps through the subject matter of the syllabus in four units aligned to the four units of the syllabus. To support the problem-solving process dictated by the syllabus, each unit is divided into four chapters that each focus on a different aspect of the problem-solving process: explore, develop, generate and evaluate (EDGE). Each chapter also includes questions, activities and projects that encourage students to move back and forth between the different phases as they explore, develop, generate and evaluate prototype digital solutions.

Teachers and students can choose to follow the sequence outlined for each unit in the book or move between the different phases of the problem-solving model, selecting activities that relate to a specific technology context and/or that support the sequence of teaching and learning relevant to their school.

Some of the subject matter required by the syllabus appears in multiple units. This duplication has been addressed in the textbook by introducing a basic overview of the concepts in Unit 1 and gradually building on this knowledge throughout the subsequent units. For example, elements and principles of visual communication, usability principles and data privacy appear in all units, but in each unit they are contextualised to support the focus of the key ideas and build on prior knowledge from previous units.

One of the challenges of writing this book was the consideration of the many possible technology contexts and programming environments that students will encounter in specific schools. The content has been presented in such a way that it can be applied to a variety of contexts and programming environments. Infobits and weblinks direct teachers and students to additional information and online activities to supplement learning, and provide a broader range of activities than could be covered in a single book.

The authors of this edition were part of the team of teachers who were selected to redevelop the syllabus. Our intent in writing this book is twofold. We want to support teachers in their implementation of the new Digital Solutions subject. We also want to support students to develop skills relevant for current, emerging and future digital solutions. We believe that this textbook encapsulates the aspirational vision and intent of the syllabus and incorporates innovative examples and case studies that will inspire teachers and students to engage with the content.

Kylie Docherty
Justin Graham
Andrew Russell

ABOUT THE AUTHORS

Kylie Docherty

Kylie Docherty has been working or teaching in the Information Technology field since graduating from Queensland University of Technology with a Computer Science degree in 1992. She has been teaching Information Technology subjects to secondary students for more than 10 years and before this worked as an IT consultant for the Queensland Government. Kylie is currently based at Mansfield State High School in Brisbane. She contributed to the development of the new Queensland Digital Solutions syllabus and also worked with other members of the expert writing team and the Queensland Curriculum and Assessment Authority (QCAA) to develop supporting resources and workshops for Queensland teachers planning to implement the new syllabus. Kylie has previously developed P–10 resources for the Australian Curriculum: Digital Technologies subject as part of the Queensland Department of Education and Training's Curriculum Into The Classroom (C2C) project, and has presented a number of Digital Technologies teacher workshops on behalf of the Queensland Society of Information Technology in Education (QSITE).

Justin Graham

Justin Graham has been working as an Information Technology leader and teacher in Queensland independent schools for the past 16 years. For most of that time, he served as Director of Information Communication Technologies and Head of Department (Technologies) at Fraser Coast Anglican College in Hervey Bay. He has been an active contributor to various Information Technology curriculum initiatives, including Queensland Lower Secondary Subject specifications, trialling and implementing the Australian Curriculum: Digital Technologies, and as an Information Processing and Technology (IPT) panellist and district panel chair. He was an member of the expert writing team for the Digital Solutions syllabus, and worked with other members of the team and QCAA to develop supporting resources and deliver workshops for Queensland teachers planning to implement the new syllabus. Justin has also been an invited speaker for Independent Schools Queensland in relation to transforming Secondary Technologies teaching departments to align to innovative and future-driven practice. Justin holds degrees of Master of Education (Leadership and Management), Bachelor of Information Technology and Bachelor of Education. He is a keen programmer in various languages and environments, including C#, Python, C-based robotics environment languages and various scripting languages.

Andrew Russell

Andrew Russell is Director of ICT at Lourdes Hill College, and has taught IPT, Information Technology Systems (ITS) and Mathematics since 1998. He has been a member of syllabus writing teams and state panels, district panel chair and on district panels for both ITS and IPT. Andrew has lectured at QUT on understanding curriculum in both IPT and ITS. Andrew has on a number of occasions presented at QSITE conferences, volunteered for CoderDojo and developed and managed learning management systems and administration databases in education. Andrew is a trained endorser for the new senior phase and currently is a passionate advocate for engaging students in the Digital Technology learning area.

IMPLEMENTATION ADVICE (FOR TEACHERS)

Learning in Digital Solutions involves developing problem-solving and thinking skills relevant to a range of educational and career opportunities. The Digital Solutions problem-solving process of exploring, developing, generating and evaluating is a new problem-based learning approach that may be unfamiliar to many teachers. The four innovative technology contexts provide flexibility to implement an engaging and relevant course of study, but teachers may find it challenging to select a specific technology context for each unit of study and establish resources and programming environments for the selected technology context. The Digital Solutions syllabus values a wide range of visual communication strategies and does not rely solely on written forms of communication. This section aims to support teachers in understanding and navigating the prescribed approaches in the Digital Solutions syllabus.

Problem-solving process

The problem-solving process used in Digital Solutions has four phases: explore, develop, generate and evaluate. The acronym 'EDGE' may help students remember the different phases of the process as they begin to learn how to solve digital problems. The first unit of the text introduces the process and the problem-based learning approach, and each subsequent unit is sequenced into four sections that align with specific phases. It is important to recognise that, despite the linear presentation of the phases, the Digital Solutions problem-solving process is non-linear. While the focus of a particular section may be on one of the phases, all four phases are revisited within each section. Icons and illustrations are used throughout the text to assist students in identifying and applying the different phases of the problem-solving process.

Thinking skills

The three thinking approaches to solving problems within Digital Solutions – computational, systems and design thinking – follow on from the Australian Curriculum: Technologies. Students will gain an in-depth understanding of when and how thinking skills should be used when teachers highlight and model their use throughout the course.

Technology contexts

The syllabus provides flexibility for schools to contextualise teaching and learning across Units 1, 2 and 3 by selecting one of four technology contexts defined in the syllabus. It is advised that schools look at elements such as programming languages and the incorporation of databases and SQL before determining which context is deployed in which unit. While many schools will adapt their course to suit one context across the four units, it is possible to incorporate complimentary contexts in different units. The text provides examples across a range of different contexts, and teachers should select activities from the text that align with the technology context that will be used for student assessment.

Algorithms

The syllabus prescribes pseudocode as the required way to document algorithms. This is particularly necessary in Units 3 and 4 as instrument-specific marking guides for internal assessment and the specifications for the external examination require it. Students who have not encountered pseudocode before may find it helpful in Unit 1 to begin with visual methods such as flowcharts or Nassi-Shneiderman diagrams alongside equivalent examples of pseudocode. By Units 3 and 4, all algorithms should be represented in the style of pseudocode recommended by the syllabus, with common keywords such as FOR, BEGIN, END, IF and WHILE. It is also recommended that other communicative elements such as indentation be modelled, taught and expected.

Programming languages

The syllabus is language-agnostic and the choice of programming language and development environment remains a school decision. Within each syllabus unit, there is a requirement to implement a procedural text-based programming language. While it is possible to use languages and development environments that also cater for other paradigms (e.g. object-oriented), the focus should remain on procedural coding structures (sequence, selection, iteration and assignment) and the ideas associated with modularisation (procedures, functions).

Programming activities in the text encourage students to explore possible programming environments or apply learning to the language and environment used at their school. Code samples are presented in a variety of different programming languages to support the language-agnostic approach of the syllabus.

Structured Query Language (SQL)

There are several online courses and tools that can be used to teach and learn SQL. It is important from the perspective of Digital Solutions in Queensland to learn generalist ANSI SQL (ANSI-92 as a minimum). While each project development platform will have variations that will be acceptable in school-based assessment, the final external assessment is likely to be based on a purely generic ANSI-style SQL. It is hence recommended during the initial learning and familiarisation stage that a learning tool is selected that exposes learners to a non-proprietary version of syntax.

Relational databases

The sample database used in Unit 2 examples is available as a NelsonNet digital asset for schools that adopt the text. Alternatively, schools may choose to use other existing datasets provided they are fit for purpose, robust and normalised to at least third normal form. Planning a fit-for-purpose relational database is beyond the scope of the syllabus and there is no specific database design methodology prescribed. Additional content for developing databases using an entity relationship modelling approach has been included in Unit 2 of the text as an optional extension.

Communicating in Digital Solutions

The syllabus has been developed to encourage the use of a wide range of communication methods and reduce the traditional focus on formal written evidence. The use of visual communication such as sketches, diagrams and mind maps is highly valued. Use of formal diagrams such as data flow diagrams (DFDs), Unified Modelling Language (UML) diagrams, web application wireframes and sitemaps is exemplified throughout the text, and the type of diagram used often depends on the technology context. When using formal diagram styles to symbolise ideas and information, students should be encouraged to follow standard conventions. The syllabus prescribes the use of DFDs and mind maps but does not preclude the use of other formal diagrams.

Unpacking units of study

UNIT 1	• Understand the problem-solving process and the importance of ongoing evaluation • Develop and refine algorithms • Use computational thinking, including decomposition and abstraction • Use design thinking to develop aesthetically pleasing user interfaces • Generate solutions using code • Evaluate solutions against useability principles
UNIT 2	• Introduce data concepts • Build on algorithmic and problem-solving skills from Unit 1 • Develop and generate structured data components • Apply normalisation processes to partially normalised datasets • Introduce Structured Query Language (SQL) • Generate and evaluate SQL solutions to data problems
UNIT 3	• Use design thinking to innovate • Use systems thinking to identify interactions and impacts • Define and communicate solutions using formal diagrams such as DFDs • Use foundational knowledge from Units 1 and 2 to generate prototype digital solutions • Apply project management principles to prepare for project-based assessment • Evaluate solutions and solution components against criteria
UNIT 4	• Build on algorithmic skills from previous units • Build on knowledge and application of SQL • Explore data exchange methods • Understand cybersecurity techniques • Evaluate social and ethical impacts

THE PROBLEM-SOLVING PROCESS (FOR STUDENTS)

Learning in Digital Solutions involves a range of problem-solving and thinking skills that apply not only to this subject but to a range of educational and career opportunities. Before embarking on this course of study, it is helpful to understand what is meant by computational, systems and design thinking skills, and how to communicate in Digital Solutions as you explore problems, develop ideas, generate components and evaluate solution prototypes. It is also helpful to have a basic understanding of how to categorise prototype digital solutions using the four technology contexts described in the Digital Solutions syllabus.

Thinking skills

There are three types of thinking skills used extensively throughout the activities in this book.

Computational thinking is used to explore a digital problem and develop ideas about how to break down the problem into small steps so that a computer can solve it. It helps to identify the inputs and outputs of a problem, create algorithms that process inputs into outputs, organise data and look for patterns in data and code that can be used to improve a digital solution. Computational thinking helps to create digital solutions that are accurate, effective and efficient.

Systems thinking is used in Digital Solutions to understand how different components of a digital solution interact with hardware, software, network, user and data systems. It helps us to analyse different approaches to solving digital problems, to identify the required components of possible solutions, and to anticipate and test possible scenarios that could occur when a digital solution is used. It also helps to analyse the social and ethical impacts of the solutions we create. Systems thinking helps to create digital solutions that are modular, reliable, economical, sustainable and secure.

Design thinking is used to imagine innovative digital solutions to everyday problems. It encourages you to alternate between thinking creatively to brainstorm crazy ideas and thinking practically to identify the best ideas that suit a particular set of criteria. Design thinking helps create innovative solutions that are easy to use, solve problems creatively and improve the user experience.

Problem-solving process

The problem-solving process used in Digital Solutions allows students to build knowledge as they solve a problem. It is a non-linear process, which means that you can move between different phases in any order. It is also an iterative process, which means that each phase is revisited often as the solution evolves. There are four phases in the problem-solving process: explore, develop, generate and evaluate. The first letters of these phases form the acronym 'EDGE'. Throughout this book, the problem-solving process is sometimes referred to as the EDGE process.



Explore

The explore phase focuses on investigating knowledge, processes and components that already exist. During the explore phase you will analyse a problem from different perspectives to understand the parts of the problem and how similar problems have been solved by existing solutions. You may need to research or learn about technical terms and concepts that relate to the problem so that you can understand it better. You may experiment with existing algorithms, data and code to understand if and how they can be applied to a solution. You may interview users, collect or search the web for data and create mind maps or concept maps that show ideas about how different resources might be created, modified or interconnected to create a solution. During the explore phase, you will identify the evaluation criteria for the solution.



Develop

During the develop phase you will brainstorm ideas and choose some ideas to refine and develop. The focus during this phase is to identify and define the requirements for new processes and components of a proposed solution and how they will interact with existing or modified components of related systems. During this phase, early ideas may be represented as sketches, mock-ups or annotated screenshots of existing solutions. Over time, the ideas will evolve and be developed into formal technical diagrams and algorithms that follow recognised conventions such as pseudocode, data flow diagrams and relational schemas.



Generate

During the generate phase, information and ideas developed and refined in the develop phase are used to build the components of a digital solution. In Units 1 and 2, the focus will be on developing and refining individual parts of a solution. In Units 3 and 4, the focus will be on developing and refining connections between components to generate an interactive prototype. During the generate phase, you will construct user interfaces, code and data, and combine them into digital solutions.



Evaluate

The evaluate phase is visited continually throughout the Digital Solutions problem-solving process. Evaluation is required throughout the explore phase to assess whether existing knowledge, processes and components have relevance to the problem under consideration. Evaluation occurs throughout the develop phase to prioritise requirements, identify good ideas, assess proposed component designs against prescribed criteria and obtain user feedback. Evaluation occurs throughout the generate phase to desk check algorithms, test programmed components of the solution and identify improvements using self-determined criteria. Evaluation involves considering a range of criteria, potential uses and impacts to determine the strengths and weaknesses of evolving versions of the prototype solution. Useability, accessibility, maintainability, reliability, security, data privacy and digital copyright are some of the criteria used to evaluate digital solutions.

Communicating in Digital Solutions

Communicating in Digital Solutions involves many different forms, modes and audiences. Written, visual and spoken forms are used to document requirements, sketch user interfaces, symbolise data interactions, interview clients and demonstrate prototypes. Some communication modes will need to be highly technical and others will need to be expressed clearly and simply in terms that are easily understood by users. Audiences will include team members, project managers, clients, customers and the general public. Communication is one of the assessable objectives in Digital Solutions, so it is important to use a range of communication strategies to demonstrate the wide range of activities that occur throughout the Digital Solutions problem-solving process. This section describes some important features of different types of communication you will use throughout this course of study.

Mind maps

Mind maps are a good way to present and organise ideas at the beginning of a project. They follow no particular rules or format apart from the convention of placing the topic word or bubble in the centre of the diagram. Words and phrases are preferred over grammatically correct sentences. Spelling is important, particularly if you are using technical terms. Similar ideas should be grouped using proximity or colour. Hierarchy can be represented with shape and size.

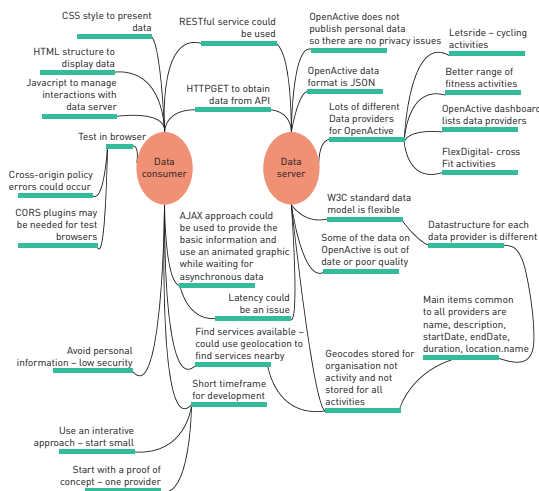


Figure 4.2.1 Brainstroming mind map for OpenActive data exchange

Mind maps are specifically mentioned in the syllabus. They are used to symbolise computational, systems and design thinking ideas and as a brainstorming tool to analyse problems and ideas at the beginning of a project. Other simple diagrams, such as concept maps and hierarchy charts, can also be helpful to identify connections between early ideas and proposed components of a digital solution. Examples of hierarchy charts and concept maps are used throughout the textbook.

Visual communication

Visual aspects of a user interface are usually communicated using sketches, mood boards, mock-ups or wireframes. Examples of these forms of visual communication are included throughout the textbook. Headings and annotations are important features of this mode of communication and can be used to identify components, explain interactions, justify decisions and evaluate impacts. Sketches and annotations should be clear, organised and logical. Colour is not usually necessary, but can be used effectively in digital mock-ups to demonstrate colour schemes.

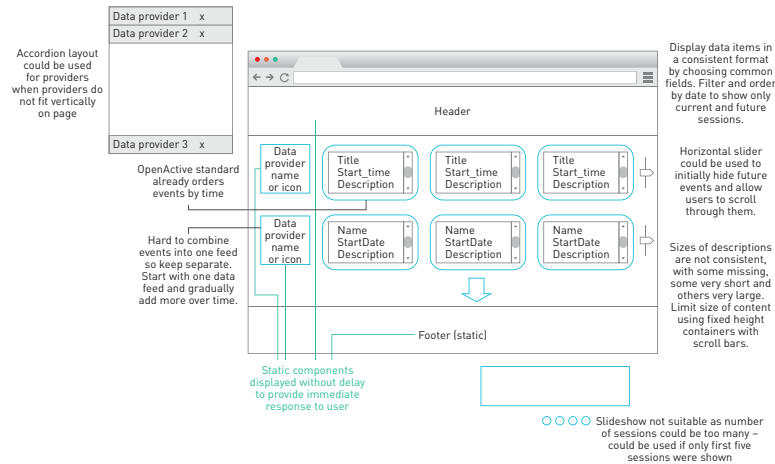


Figure 4.2.4 Static and dynamic elements of the OpenActive data exchange interface

The visual communication elements and principles explained throughout the textbook should be practised when preparing mock-ups and wireframes of user interfaces. Annotations explaining how the elements and principles have been applied are a good way to demonstrate understanding of the importance of good visual communication in both the documentation and components of the digital solution.

Data flow diagrams

Data flow diagrams (DFDs) are technical diagrams. They have a specific format and rules that need to be followed. They consist of entities, data stores, processes and connections that have specific shapes and naming standards. Specific rules must be followed to create and label connections between the shapes on the diagram. DFDs and the rules that must be followed to create them are explained in Units 2 and 3.

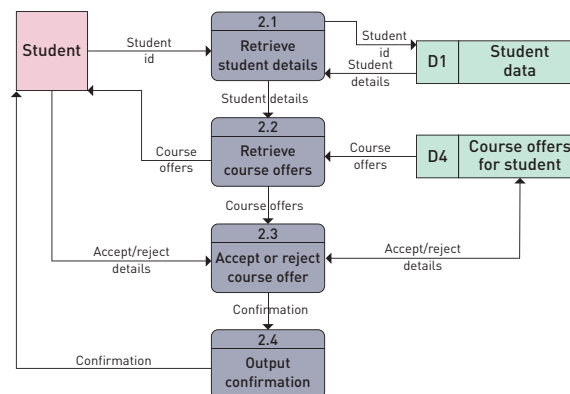


Figure 2.2.4 Data flow diagram for student course offering review and acceptance

DFDs are the only type of formal diagram specifically mentioned in the syllabus, but other types of technical diagrams can also be helpful in defining the requirements of a digital solution. UML object diagrams, interaction diagrams, ER diagrams and relational schemas are other types of technical diagrams that can be used to symbolise and explain the components of digital solutions. These diagrams are used throughout the textbook, and online resources are available to explain the specific rules that apply to them.

Pseudocode

Algorithms should be expressed as structured pseudocode that follows specific conventions. It is important that pseudocode is sequenced in a logical order, independent of any coding language. Unit 1 explains the conventions of communicating algorithms in pseudocode form.

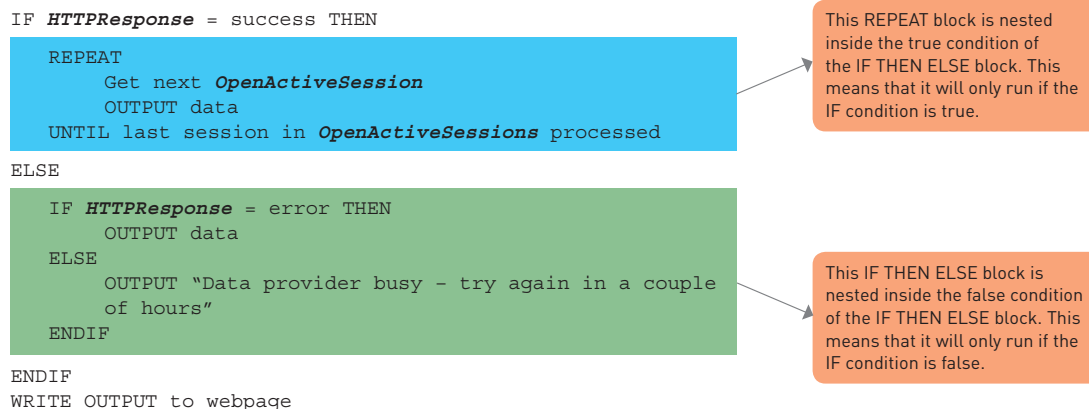


Figure 4.2.7 Complex algorithms can be constructed by nesting one control structure within another control structure.

Flowcharts and input-process-output tables are not prescribed by the syllabus but are used in Unit 1 to introduce the concept of algorithms, and as a planning tool to identify data inputs and outputs. Pseudocode is the preferred communication method for all algorithms in Digital Solutions.

Comment syntax

Comment syntax is a powerful communication tool in Digital Solutions. It can be used to explain the purpose of coded components or functions, demonstrate understanding of complex code structure and improve the maintainability and readability of a program module.

```

//Set the url to the FlexDigital API
var url = 'https://open.flexidigital.co.uk/oa/';

//Send a request for the data
jQuery.getJSON(url, function(data) {
    /*****
    Define the callback function
    This function will be used to process the HTTP Response
    Purpose: To display the activity session data in the appropriate section
    *****/
    //Set up a pointer to the section of the page
    //where the dynamic data will be added
    var section = document.querySelector('section');
  
```

Comment syntax is sometimes referred to as internal documentation because it is only seen by the program developers. Code comments are not processed by the computer or presented to the user but are helpful in communicating the computational thinking and algorithmic approaches used in a segment of code.

Other forms of communication

Written, oral, audio and video forms of communication can also be used in Digital Solutions to write technical proposals, record interviews, show evidence of testing and demonstrate the functionality of a prototype. In written communication, spelling, grammar and paragraph structure are important to ensure that what you are communicating is clearly understood. In oral presentations, visual aids, diction, eye contact, intonation and volume are important to ensure that the audience understands and maintains interest in what you are saying. When making audio recordings, it is important to eliminate any background noise and play back the recording to make sure it is loud enough. Video recordings are a combination of all the other forms and require error-free visual aids, clear diction, appropriate eye contact and minimal background noise. Video presentations can be used effectively in Digital Solutions to record the interactions of a prototype digital solution and demonstrate its functionality. Voice-overs can be added to explain the processes used to create the solution, and reduce the need for extensive written documentation.

Technology contexts

There are four technology contexts described in the Digital Solutions syllabus: web applications, mobile applications, interactive media and intelligent systems. Technology contexts provide a theme for the activities in a particular unit, and not every technology context needs to be covered. It is the teacher (not the student) who selects the technology context for each unit. Some activities and projects from the textbook apply to specific technology contexts. The table below categorises the different types of digital solutions students may create.

TECHNOLOGY CONTEXT	DESCRIPTION
Web applications	Applications that use a web browser as the user interface
Mobile applications	Mobile phone or tablet applications
Interactive media	Animations, simulations, games, learning objects or tools
Intelligent systems	Wearables, Internet of Things (IoT) systems, robotics or automation software

Unit

1

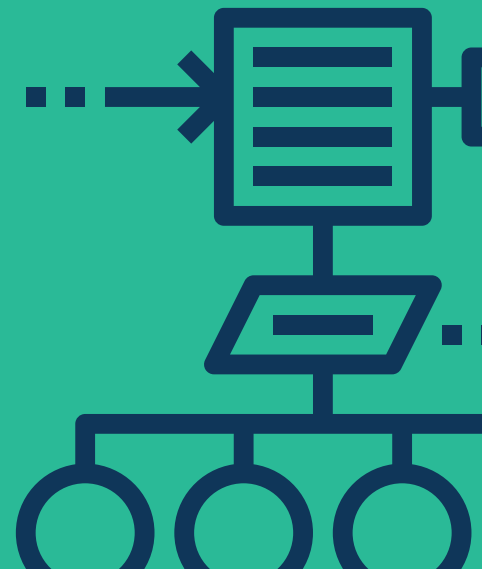
CREATING
WITH CODE

INTRODUCTION

Unit 1 explores the first unit of the QCAA Digital Solutions syllabus, which focuses on the creative and technical aspects of developing interactive digital solutions and is divided into four topics: section 1.1 Exploring digital problems; section 1.2 User experiences and interfaces; section 1.3 Algorithms and programming techniques; and section 1.4 Programmed solutions. Students apply the four phases of the Digital Solutions problem-solving process – explore, develop, generate and evaluate (EDGE) – to study algorithms, programming features and useability principles to generate small interactive solutions using programming tools, and gain a practical understanding of programming features.

KEY IDEAS

- Programming elements
- Components of digital solutions
- User interface
- Useability principles
- Programming problems
- User experience
- Programming requirements
- Prescribed criteria
- Self-determined criteria
- Impacts



1.1

EXPLORING DIGITAL PROBLEMS

In this section, students unpack and use **computational**, **design** and **systems thinking** when exploring digital problems. Students explore digital systems and develop ideas that lay the foundations for creating with code. Students learn how to break a digital problem down into parts, understand the components and how they contribute to the digital system. While exploring the digital problem, students will identify system functionalities, scope and constraints while considering the user experience and requirements. Students use mind maps to unpack digital problems and explore existing and new ideas. Students explore feasibility and personal, social and economic impacts of a digital solution. Students explore and develop ideas in a **web** or **mobile application** context, interactive media context or an **intelligent systems** context. Students develop evaluation criteria to measure success and gain an understanding of where and how digital solutions are used to meet user needs.

What is a digital problem?

A digital problem can have many parts that need unpacking and understanding. This is true of many types of problems; what makes a digital problem different, however, is that its solution consists of digital hardware and software working together to form a digital system. In a digital system, data goes in (input), is processed (process) and is sent out (output) to achieve a solution. Humans can control digital systems through the use of **algorithms**. This type of solution calls for a unique set of problem-solving skills, which will be explored in Unit 1.

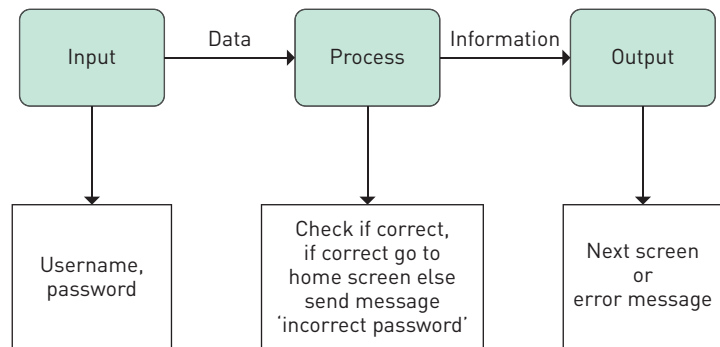


Figure 1.1.1 Software decomposed into input, process, output

Students need, firstly, to explore the hardware systems and software systems involved, and then how the input, process and output (IPO) work together. The third component is exploring user needs. This is the starting process for developing a digital solution.

An example of a software subsystem is a typical login screen, where the user needs the correct username and password to be granted access. This could be viewed as input, process and output as illustrated in figure 1.1.1.

We can also view this in an IPO chart, as shown in table 1.1.1.

Table 1.1.1 Software IPO chart of a login system

Input	Process	Output
Username Password	IF ok THEN go to home screen ELSE send message "incorrect password" ENDIF	Next screen or Error message

Solving digital problems

When faced with a digital problem to solve, you will likely start in a linear fashion as you **explore**, **develop**, **generate** and **evaluate** (EDGE). You will quickly discover that this is not a linear process, and there is a continual need to revisit ideas and further explore the problem when developing and generating digital solutions. A concise and non-exhaustive summary of the four problem-solving phases is set out in pages x–xi. The benefits of using this approach become more evident when you apply it to solving a problem.

Below is a brief explanation of how exploring, developing, generating and evaluating can solve digital problems. This will help you understand how these problem-solving phases can be used within Unit 1 and beyond.

Exploring

To **explore** a digital problem:

- 1 **Restate the problem** into concise bullet points.
- 2 **Explore a human need or want.** The more you understand the needs and wants of the users, the more successful your solution is likely to be. Ways to explore human needs or wants can involve:
 - researching existing solutions
 - establishing who the target audience is (user **personas**)
 - role-playing from the user's perspective to understand behaviours and how they might interact with your solution
 - collecting feasibility feedback data from a sample population. For example, a feasibility feedback survey of Shazam, a mobile app-based music identification service, might look like this:

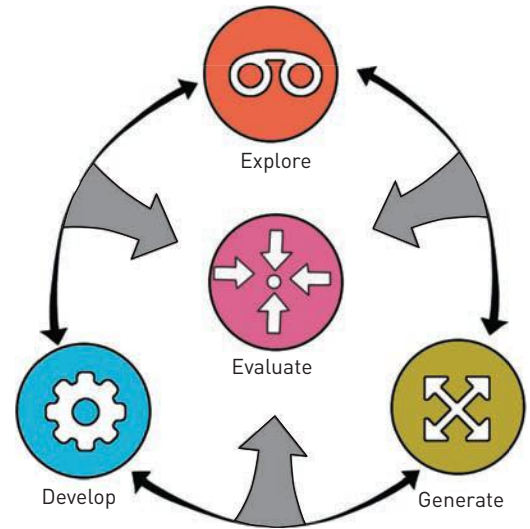


Figure 1.1.2 Digital Solutions problem-solving process

ACTIVITY 1.1.1

SHAZAM FEASIBILITY FEEDBACK SURVEY

The goal of this survey is to find a group of people that can give you information about your idea/product/market opportunity.

- Do you enjoy listening to music (yes or no)?
- How often do you find yourself wondering about the name of a song that you hear (1 = never, 10 = all the time)?
- How can we contact you to talk further about this?

Resulting metrics (data we received)

Divide the number of people who responded in a selected way by the number of people who took the survey.

Results

45/50 said yes, they enjoy listening to music, 22/50 reported a 5 or greater about not knowing names of songs, 40/50 included at least one way to contact them. What we learned is that there is a large market size for our product, most people wonder about the name of a song they hear, people are interested in talking to us about it.

- 3 **Decompose** (break down) the digital problem into smaller, more solvable subsystems. This can be done with a mind map or an annotated sketch, as illustrated in figure 1.1.3. It is much easier to explore new and innovative ideas if a problem has been broken down into smaller parts. This also helps you communicate an understanding of the system and how the subsystems work together in a digital solution.

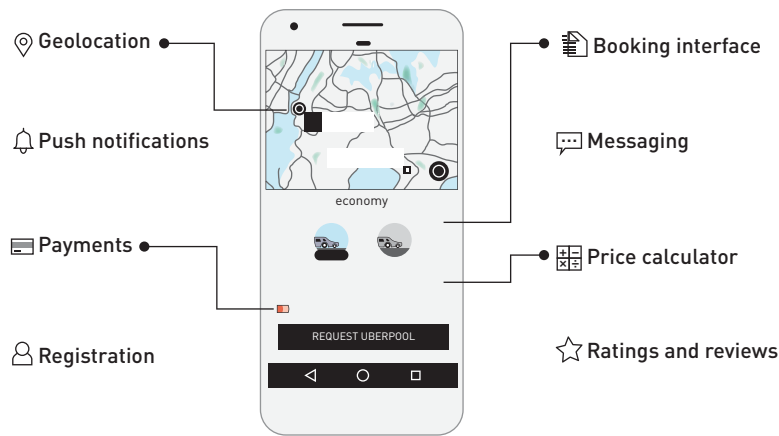


Figure 1.1.3 Decomposition of a smartphone app

Abstraction

Figure 1.1.3 highlights how subsystems are hidden from the user. This is known as abstraction. It is a computational thinking skill you will learn and apply in this section. Abstraction is used to hide complexity so you can focus on one part of the problem at a time. For example, users of flight simulator mobile apps want to 'tilt' their phone to turn the plane and are not interested in the internal phone systems. The simulator app is easier to use when complex systems are hidden from the user.

ACTIVITY 1.1.2

Explore your smartphone home screen. Sketch your home screen on paper and complete a basic decomposition of the screen. Identify subsystems that are abstracted from the user and, as far as possible, identify the hardware and software components within each subsystem. Refer to figure 1.1.3 as an example of a solution of this activity.

- *Impacts and disruptions.* What possible impacts and disruptions could your solution have on society – for example, on the environment, on the economy, on security and privacy?
- *Scope and limitations.* What scope and limitations are there that relate to the process and the end product? It is important not to be distracted by unrelated external events.
- *Knowledge, skills and tools.* Explore any knowledge, tools and skills that might be needed to get the job done. In this unit we will focus on generating digital solutions through programming. The following is a non-exhaustive list of some typical questions to be explored when starting a new project:
 - What software is needed to develop and generate a user interface?
 - How will designs be generated?
 - What software and programming language is needed to make an interactive user interface?
 - How will the digital solution be tested?
 - What programming skills are needed?
 - What devices and operating systems will the solution run on?



Developing

Develop and formalise your ideas. In Unit 1, this means taking your sketches and mind maps and producing more formalised, technical diagrams, such as flowcharts, to demonstrate a more detailed view of the system and subsystems that are abstracted from the user. An example is the use of wireframes that demonstrate the prototype layout of user interface (UI) designs, with dimensions and algorithms for the functionality (processes) within subsystems, in your digital solution. Your solution might be a robotic vacuum cleaner, a solar panel cleaning drone, a mobile app that connects students with other students who can mentor them in subjects and homework, or a smart umbrella that lets you know when it is too dangerous not to be under cover. The develop stage should result in more detailed and technical sketches that visualise your design, computational and system ideas from the explore phase. In Units 1 and 2 you will focus on one part of a solution. Unit 3 explains in detail some of the types of diagrams you can use in the develop stage to communicate more complex technical ideas.



Generating

Generating means constructing your final solution by programming code and creating interfaces using the tools you have selected to get the job done. Theoretically, if the explore and develop phases are completed really well, the generation should be a trivial task. However, it is typically the case in digital solution production that you will revisit

the explore, develop and evaluate phases while working in the generate phase. Generating involves documenting your solution in the form of comments and highly structured, efficient and meaningful code.

Evaluating

The evaluate stage is the most powerful and continuous part of the problem-solving process, as illustrated in figure 1.1.2. In Unit 1, part of your evaluation involves debugging code by continually testing and refining components throughout the whole project. When refining the solution, you should keep asking, 'Is my digital solution effective and efficient and, most importantly, meeting the original requirements?' Criteria checklists are commonplace in evaluation as they are handy tools for testing, tracking and checking progress. The use of an activity log or journal is a good way to document your evaluation process. Continue to refine by testing and debugging code, which is further explored in Unit 1.

Knowledge and skills used in the information technology world can be grouped into three ways of thinking: design thinking, computational thinking and systems thinking. As you journey through this book, you will learn to use these three thinking skills in different technology contexts to generate digital solutions.



Evaluate

Using mind maps to explore digital problems

Let's apply the first part of the problem-solving process to understand the following situation:

A non-tech savvy shop owner notices many people walking past the store because they are listening to songs on their phone while messaging friends or playing their favourite game. If only the device they are giving their undivided attention to could alert them of special offers that they are walking past! The shop owner hires people to stand outside with microphones and shake signs to entice people in, but people are absorbed by their devices. By using geolocation sensors and apps like Google, for example, it is possible to alert users of massive savings in the shop. The shop owner knows nothing about mobile technology. What can you do to help the shop owner?

EXPLORING THE PROBLEM

- 1 Restate the problem succinctly.
- 2 Analyse the human need or want.
- 3 Decompose the problem into smaller parts.
- 4 Identify requirements.
- 5 Recognise constraints.



Explore

- 1 Restate the problem succinctly. Here a bullet point list can help you to be concise and succinct:
 - Shop owner wonders if technology can help alert people walking past about sales in the shop.
 - Many people are using and engaged with their smart devices as they walk past the shop.
- 2 Collecting data is the best way to **analyse** human needs and wants. In this situation, there are two areas for further exploration into human needs and wants. The first is the shop owner, who wants to use technology to target potential customers walking past and alert them of sales. The second is the user persona of the target audience – the customers. By understanding factors such as the age group and gender of the target audience, the technology they use, and apps commonly used by that age group, you can design and tailor a user-oriented product or service that meets their needs. In this case, the target group listen to music, mostly use smartphones and are aged between 20 and 30. Put simply, the shop owner needs a way to attract the attention of people walking past the shop. What do they all have in common? Using key persona commonalities would be a way for the shop owner to alert people to the bargains they are walking past. Unit 3 in this book covers this in further detail.

- 3 Break the digital problem down into smaller logical components and brainstorm new ideas. Figure 1.1.4 demonstrates the first-level exploration of the shop owner problem. In this case, we have decomposed the problem into three groups:

- device being used – hardware, software, inputs, outputs
- user persona/behaviours (from step 2)
- functionality (processes).

Figure 1.1.4 shows the first version of the mind map for this problem. As the solution evolves through other phases of the problem-solving process, the developer will revisit the explore phase. Later versions of the mind map may include sketches of similar solutions, possible algorithms and techniques, or links to relevant tutorials and resources.

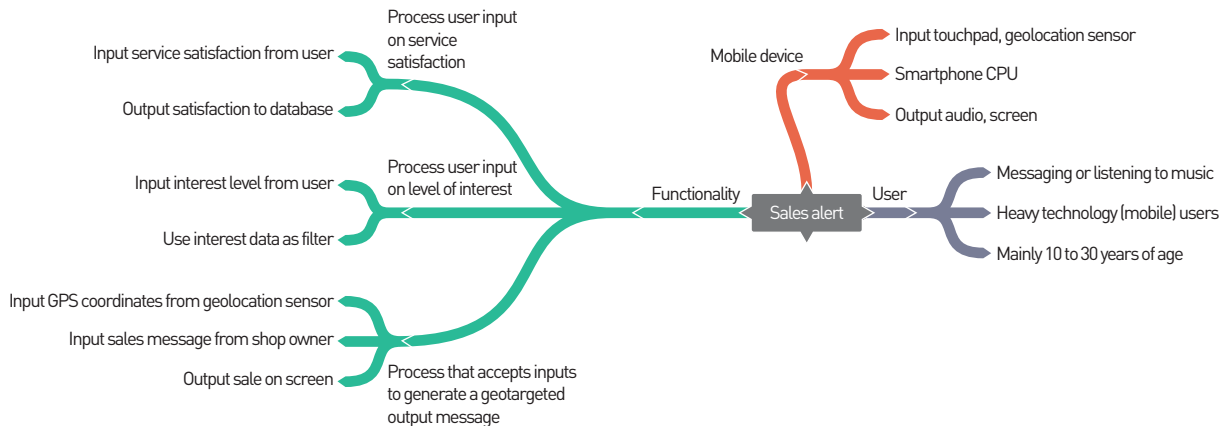


Figure 1.1.4 Mind map used for brainstorming

- 4 Identify the requirements. Based on figure 1.1.4, some initial requirements could be:
- use geolocation to find nearby stores
 - allow shop owners to broadcast sales
 - allow users to choose what type of sales they are interested in
 - allow users to provide feedback about sales.
- 5 Recognise and consider the personal, social and economic constraints of this technology. Will people feel privacy and personal space has been breached with pervading advertisements and tracking via geolocation? Are there legal restrictions on geo-targeting? Is it profitable?

Geo-targeting is the practice of delivering content to a consumer – via mobile or the Internet – using geographic location information about that individual. At a basic level, a business can restrict its reach to consumers only located in a defined geographic area. Location often provides much deeper, more meaningful and identifiable traits that tell you what a person wants, needs or is interested in.



ACTIVITY 1.1.3

Explore how Domino's Pizza company uses geo-targeted text message marketing. Use the Internet to research and gather information to help you answer the following.

- 1 Identify and list human needs or wants relevant to this case.
- 2 List the personal, social and economic impacts of this practice.



1.1.1 Digital technologies hub: systems thinking

Systems thinking when exploring digital problems

As illustrated in figure 1.1.4, exploring digital problems requires systems thinking in terms of the hardware (device) and software (functionality) components and understanding how they work together to form a digital solution. The Australian Curriculum defines systems thinking as '[a] holistic approach to the identification and solving of problems, where parts and components of a system, their interactions and interrelationships are analysed individually to see how they influence the functioning of the whole system.'

Decomposing hardware and software of a digital solution

Systems thinking involves wanting to know what is happening beyond the user level – for example, the mechanical system of a car that makes the car move, or how the sensors work in your smartphone to make it do something ‘smart’.

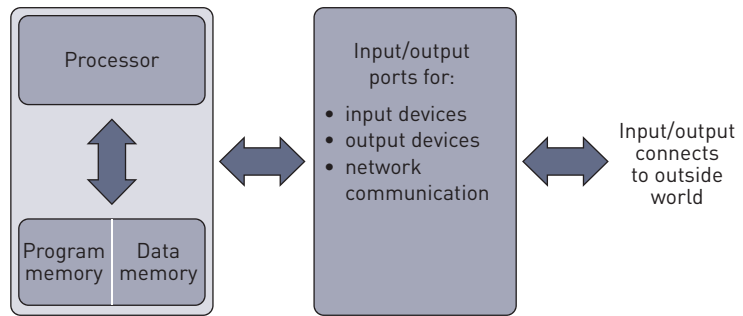


Figure 1.1.5 Hardware components of a digital solution

CASE STUDY

Google Home and Amazon’s Alexa are examples of home automation technologies. When you say ‘Hello Google’, ‘Siri’, ‘Hey Cortana’ or ‘Alexa’, this voice command is received as input by the smart device. A microprocessor controlled by software processes the voice input and generates a digital message that can be transmitted to control other digital systems. It is not necessary to understand all the details of the **protocols** used to transmit data, but identifying that this subsystem exists helps us to recognise constraints that limit the type of device that can be controlled, and how far away it can be moved before the system fails. You will learn more about data exchange systems and methods for sending messages between digital components in Unit 4. In the past, digital systems could only be connected using cables, but wireless protocols now allow digital signals to be sent over the airways to nearby devices. Wireless communication protocols such as wi-fi and Bluetooth are well known, but new protocols are sometimes needed to overcome limitations of existing ones. The Zigbee protocol is an emerging wireless protocol that combines the low energy requirements of Bluetooth and the longer range of wi-fi. It is becoming popular for Internet of Things (IoT) projects and is used in many medical, industrial and home automation systems.

Figure 1.1.5 illustrates a simple decomposition of a digital system. These are the fundamental hardware components of all digital solutions:

- *processor* (CPU)
- *program memory* (RAM)
- *data memory* (RAM and hard disk)
- *input/output* devices.

The hardware system is controlled by software loaded in program memory. Software is the coded form of an algorithm. Developing algorithms and generating software components is explored in sections 1.3 and 1.4.

Hardware capabilities are advancing at an unprecedented rate, which provides software developers with new opportunities to create innovative digital solutions – the focus of Unit 3. Some people predict that the next decade will be ‘the decade of disruption’. Typically, an innovation will be known as disruptive if it changes the way we live – for example, a smart light bulb that turns itself on and off. This is an example of a digital solution to an everyday interaction we have with our environment.

A more detailed hardware view of a programmable board such as the BBC Microbit is shown in figure 1.1.7 and identifies sensors such as an accelerometer and a compass as input. Using a programmable board, you can develop software that processes sensor data to perform many tasks – such as security alarms for a ‘smart home’, an activity tracker that maps your walk, and many other interesting applications.

Raspberry Pi, Arduino, Microbits, mobile phones, tablets, laptops and many more devices that we know today as smart devices can be decomposed into input, output and processing subsystems.



Using systems thinking, software can also be decomposed into basic components. The basic components of software are:

- input data (from users or other systems)
- process (algorithm)
- output data (such as messages, images, sound, etc.).

These input, process and output components can be represented in an IPO chart, as shown in Table 1.1.2. Software gives hardware devices a behaviour or function. Without software the device is just an inanimate circuit board.

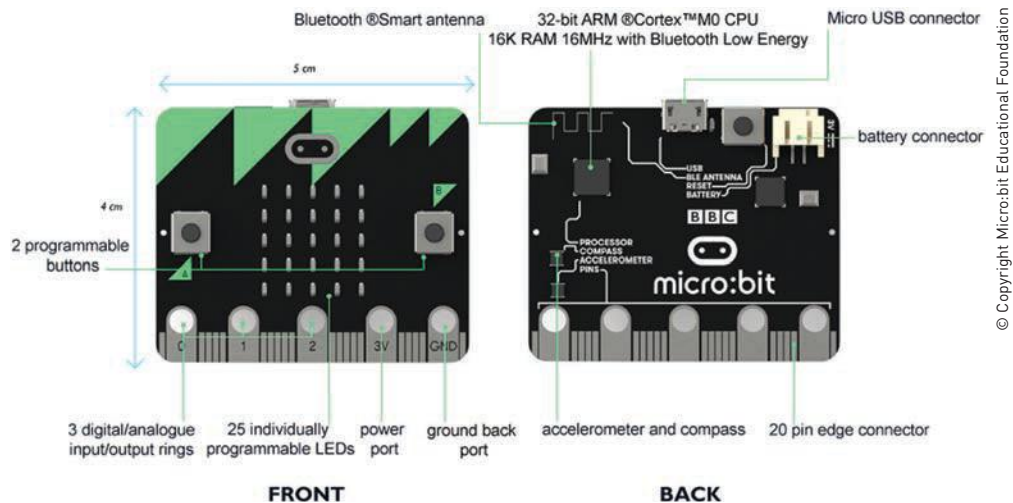


Figure 1.1.6 BBC Microbit

Table 1.1.2 Software IPO chart

Input	Process	Output
Voice command Device status (on or off)	REPEAT LISTEN for voice command UNTIL voice data input received IF voice data input matches "Turn TV On command" THEN IF Device status is on THEN SAY "TV is already on" ELSE SEND PowerOn command to device SAY "TV is now on" ENDIF ENDIF	Audio message, e.g. 'TV is now on' Digital transmission, e.g. power on

ACTIVITY 1.1.4

- 1 Decompose an IoT device in terms of input, process and output.
- 2 Explore an example of a digital solution and develop a mind map similar to that shown in figure 1.1.4. In your mind map, include a functionality branch (application), a user persona branch, and the device branch (hardware) that represents the system.
- 3 Identify an example of a disruptive technology. Explore how it has had an impact on society.
- 4 Explain in your own words what systems thinking is.
- 5 Explain the difference between hardware and software in a digital solution.
- 6 What is an IPO chart and what is it used for?

Computational thinking when exploring digital problems

Computational thinking is a collection of six key mental strategies (listed below) that are involved in formulating a problem and expressing its solution in such a way that a computer can effectively carry it out. Computational thinking includes:

- *using logic* – imposing a set of rules of right (correct) reasoning on our thinking. In philosophy, logic is the science of studying arguments, which are generated using logical statements. A computer program (algorithm) is also generated from a sequence of logical statements. Logic also uses truth statements such as ‘The sky is blue’. This statement is either true or false and the outcome of the statement influences the logical argument. This is not necessarily a natural way of thinking, and it takes practice. Logical thinking is inappropriate in some contexts, such as in the arts or in romance, but it is ideal for computer programming or mathematics
- *evaluating* – the process that allows us to make sure our solution does the job it has been designed to do and to think about how it could be improved. Failure to evaluate can make it difficult to write a successful program
- *thinking algorithmically* – the process of moving towards a solution through the clear definition of the steps needed
- *identifying patterns* – which is about observing patterns, trends and regularities in data
- *decomposing* – breaking down data, processes or problems into smaller, manageable parts
- *abstracting* – hiding detail, which leads to an elegant and simple view of things.

Section 1.3 focuses heavily on applying computational thinking to develop algorithm ideas.



Adapted from Barefoot Computing at School, UK, poster, <https://barefootcas.org.uk>

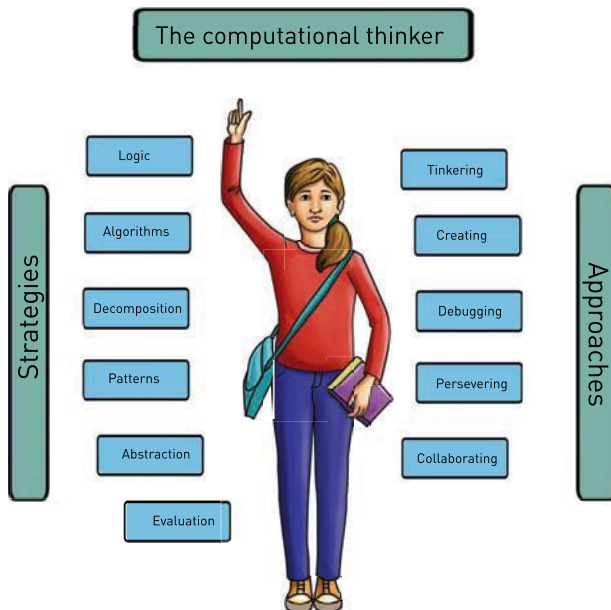


Figure 1.1.7 Summary of computational thinking steps and the approaches used to create digital solutions

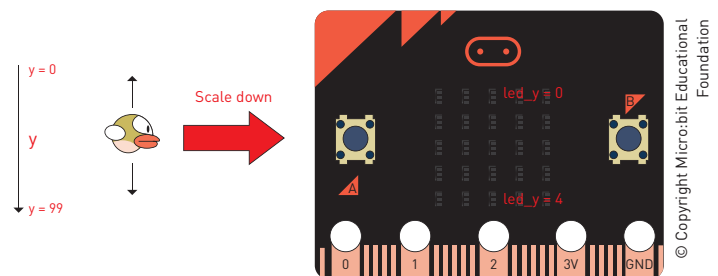


Figure 1.1.8 LED coordinates illustrate how algorithms can be used to control a physical system in a game such as Flappy Bit.

Table 1.1.3 Flappy Bit IPO

Input (data)	Process (algorithm)	Output (action)
Microbit tilted up	SET <code>led_y</code> TO scaled <code>y</code> SUBTRACT 1 FROM <code>led_y</code> DISPLAY flappyBit at position <code>led_y</code>	LED display bit for flappy bit moves up



In figure 1.1.9 and table 1.1.3, the following components of computational thinking were used:

- decomposition – to create the IPO chart in table 1.1.3, breaking down the software component of the digital solution into inputs, process and outputs
- algorithmic thinking – to create a step-by-step process a computer can follow
- logic thinking – to identify possible inputs
- evaluation – continuously to verify that it works as expected.

It is quite normal to use computational thinking only on smaller subsystems. Developers use abstraction to focus on one part of the solution at a time. This will become more evident in section 1.3, where modularisation is used and the code is broken down into smaller modules. The outputs from one part often become inputs to another part.

ACTIVITY 1.1.5

Computational thinking involves identifying patterns. A stack computer is loaded with boxes from a conveyor belt, as demonstrated in figure 1.1.10. The boxes are marked with a number or an operator (+, −, * or /). The computer is loaded until the top box is a box marked with an operator. This operator is then used on the two boxes below it. The three boxes are then fused into one single box and marked with the outcome of the calculation. In the stack computer, calculations are entered in a different way to a normal calculator.

- 1 How should the following computation be entered: $4 * (8 + 3) - 2$?
- 2 What would an IPO chart look like for this activity?

4 + 6 must be
entered as 4 6
+

5 − 4 must be
entered as 5 4
−

7 * 4 + 6 must be
entered as 7 4
* 6 +

7 + 4 * 6 must be
entered as 7 4
6 * +

(2 − 4) * (6 + 9)
must be
entered as 2 4
− 6 9 + *

Figure 1.1.9 Stack computer

Design thinking when exploring digital problems

Design thinking refers to the creative strategies for visualising and generating a range of creative and innovative ideas (divergent thinking), and analysing and evaluating those ideas that best meet the criteria for success (convergent thinking). Design thinking influences all aspects of a digital solution, from the way a web page or mobile app looks, to the way an algorithm or a robot is constructed. You need to explore creatively and ‘tinker’ with technology without the fear of breaking something.

‘Anyone who has never made a mistake has never tried anything new.’

Albert Einstein

Remember the shop owner who wants to use technology to alert people walking past their shop about sales. When exploring this problem, divergent thinking was used to brainstorm new ideas and add them to the mind map, for example ‘input service satisfaction from user’. Convergent thinking was used to evaluate the

ideas and decide which ideas should become required features of the application – for example, ‘allow users to provide feedback about sales’. Section 1.2 looks further at design thinking and using elements and principles of visual communication to develop effective user interfaces. Throughout the book there are examples and challenges for applying design thinking in problem-solving strategies.

A sandbox is an example of a playground for programmers with new ideas to practise their design and computational thinking. A sandbox is an isolated computing environment in which a program or file can be executed without affecting the application in which it runs. Sandboxes are used by software developers to test new programming ideas.



Combining design, computational and systems thinking when exploring digital problems

Getty Images/John Lund



Figure 1.1.10 Face map

Design thinking is quite often used in conjunction with computational and systems thinking – for example, in many countries, cameras on the streets use facial recognition technology as a security measure to help prevent crimes. A person seen committing a crime and under investigation can be tracked in reverse from the location of the crime. Someone witnessing a crime can call in, at which point the footage from that and nearby cameras are scanned. An example of design thinking in the development of this technology was the mapping of the face, shown in figure 1.1.11. Computational thinking was required to develop the algorithms and generate code for the facial recognition software. Engineering the massive computing power and hardware needed to link 160 000 cameras as inputs to the software processes and outputs required systems and design thinking.

ACTIVITY 1.1.6

Below is an example of logic used in a face recognition library called clmtrackr. These are the points in the array returned by `ctracker.getCurrentPosition()`.

```
var facepartX = positions[23][0] < X < positions[25][0];
var facepartY = positions[24][1] < Y < positions[26][1];
```

Look up each number used in the code above on the numbered face model image. Can you guess what part of the face is being tracked by the X and Y coordinates in the code?

Based on your logic from the previous question, what numbers would you expect to see in the code for tracking the mouth?



CASE STUDIES

The following three case studies will reinforce students' understanding of how to explore digital problems. The first uses a web context; the second, an interactive media context; and the third, an intelligent systems context. The skills covered so far will be further explored, adding new tools and skills.

WEB APPLICATION: RUSTY AND THE EZIBEATS

Rusty and the Ezibeats are a contemporary rock band. They have approached you to make a prototype responsive front-end website. They have a youthful following of mostly mobile device users. They want to attract this 'tech generation' with a website that has a home page that includes fading images,



Explore

information about the band, tour dates including ticket purchase, contact information and merchandise. The home page should also include a purpose-built, easy-to-use online forum, where fans can use a 'post' button to publish information about concerts and the band. This forum also gives the band an opportunity to collect data on their faithful followers. When fans sign up to the forum they need to create a username and password. The band would also like to collect data on the age, gender and country of origin of users to help them with marketing strategies. In the forum, the page needs to welcome the user by name.

Explore the problem

- 1 Understand the constituents of a digital problem by restating and summarising:
 - Band name 'Rusty and the Ezibeats'
 - Responsive front-end website (no data coding)
 - Pages: Home, Band, Tour dates including ticket purchase, Contact information, Merchandise, Forum and sign-in for forum
 - Target audience are youthful and enjoy rock music
 - An online forum that provides feedback from fans and greets the user
 - A login that captures membership numbers and some data about the user
- 2 Analyse the human need or want. In this example, the Ezibeats have already analysed their target audience as youthful and predominantly mobile phone users. They have asked for a responsive website, which means that web pages will be suitable for a range of screen sizes, browsers and devices.
- 3 Decompose the problem by breaking the problem down into parts, for example:
 - Create a mind map.
 - Break the problem down into three different views: *Functionality* (code); *Web pages* (HTML); *User interface*. Figure 1.1.12 shows how a mind map can be used to decompose the Ezibeats website problem. A numbering system is used to match each process to a web page.

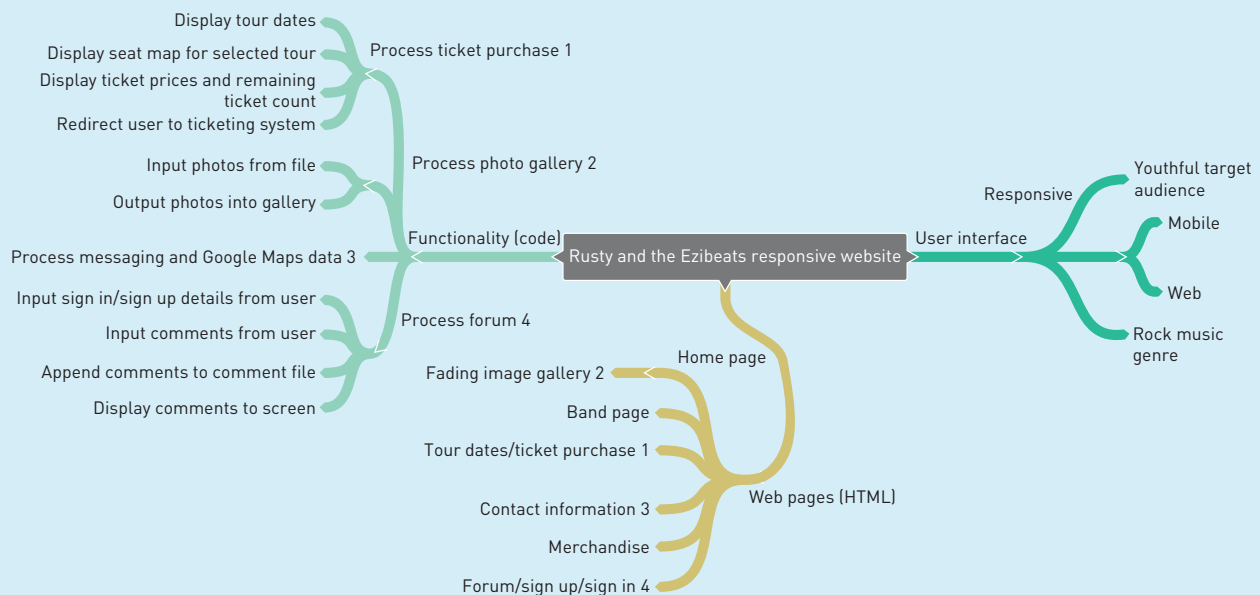


Figure 1.1.11 Mind map

- 4 Identify essential elements, components and features of the digital solution (requirements):
Figure 1.1.13 develops the ideas further into a 'site map' or 'hierarchy chart', a graphical representation of the top-down organisation or structure of a solution. The benefit of this is in showing sub-pages or modules that flow from a parent page or module. A possible site map for the Ezibeats web application is shown in figure 1.1.13.

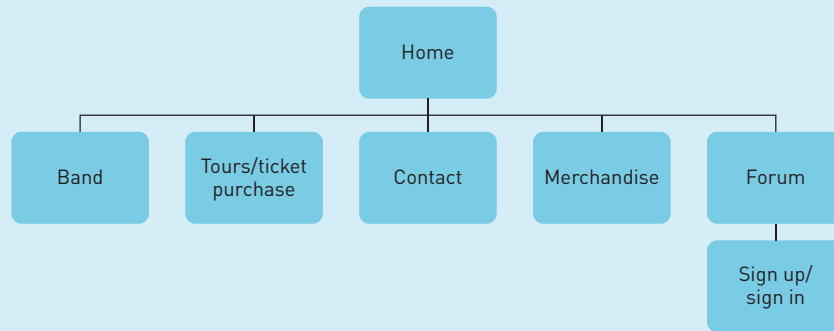


Figure 1.1.12 Hierarchy chart

- Home page – navigation bar, fading image gallery (algorithm)
- Band page – band member profiles (photo of each member) and history
- Tours/ticket sales – show upcoming tours and available tickets for purchase (algorithm)
- Contact – ‘Contact us’ page, Google Map location (algorithm)
- Merchandise – shopping cart algorithm (future project)
- Forum – sign in/sign up (algorithm)

The mind map has highlighted four processes (functionality), which will eventually be generated into JavaScript. In this development stage, we can use IPO charts to indicate the inputs, process and outputs:

Process Ticket purchase 1 IPO chart

Input	Processes	Output
Tour dates from file EventName EventDate No. of tickets	Algorithm to process ticket data	Redirection to appropriate tour page in ticketing system

Process Photo gallery 2 IPO chart

Input	Processes	Output
Images from a file	Algorithm to process slide show	Photos dynamically fade in and out based on a set interval

Process messaging and Google Map 3 IPO chart

Input	Processes	Output
Name Email Message Send button GPS location	Algorithm to process message Algorithm to process location on map	Email message text Map with location for how to find us

Process Forum 4 IPO chart

Input	Processes	Output
Username Password Click submit button Comments Click post button	Algorithm to collect username and check password	Gain entry to forum or Warning prompt of incorrect username Comments posted



Wireframes are a tool used to develop layout concepts, navigation and technical details such as HTML elements, CSS style rules and page functionality (JavaScript) for the user interface. This website requires a responsive design, therefore wireframes are needed for both desktop and mobile. This is explored further in section 1.2, User experiences and interfaces.

5 Recognise constraints and limitations of the environment:

- Project timelines, developer knowledge and number of developers will impact the scope of the project. Not all pages and features will be available in the first version of the prototype.
- Databases are not introduced until Unit 2 and data exchange processes are not introduced until Unit 4, so some strategies for accessing tour dates and securing passwords are beyond the scope of Unit 1 projects. The prototype solution will be limited by the functionality that can be provided using local storage options.

INTERACTIVE MEDIA: ASCII ALIENS

You are a developer working for an educational company. The company wants to use the famous Space Invaders arcade video game, created by Tomohiro Nishikado in 1978, to help reinforce students' understanding of the ASCII table and the binary conversion to characters used by computers. You are to create a game that is playable in a web browser. You need to follow the Space Invaders theme, and use the same gaming approach of defeating the aliens before you are destroyed and increasing difficulty with each level, but come up with your own catchy name and characters.

Explore the problem

1 Restate the problem succinctly:

- Create a web-based game that follows the Space Invaders format of 'shoot to survive'.
- Reinforce students' understanding of the role of the ASCII table in conversion from binary machine code to human-readable characters.
- Must be playable in a web browser.
- Catchy, with own characters that match the context of a binary conversion game.
- Level of difficulty increases.
- Assume keyboard controls and desktop.

2 Analyse the human need or want:

- Research ASCII to understand the conversion process. We use ASCII in the process of binary-to-text encoding.

For example, when a key such as the letter A on the keyboard is pressed, it is not interpreted as an 'A' but as a binary number. The binary number represents an ASCII code. Each ASCII code maps to a specific key on the keyboard. Figure 1.1.14 shows the ASCII codes displayed on a keyboard.

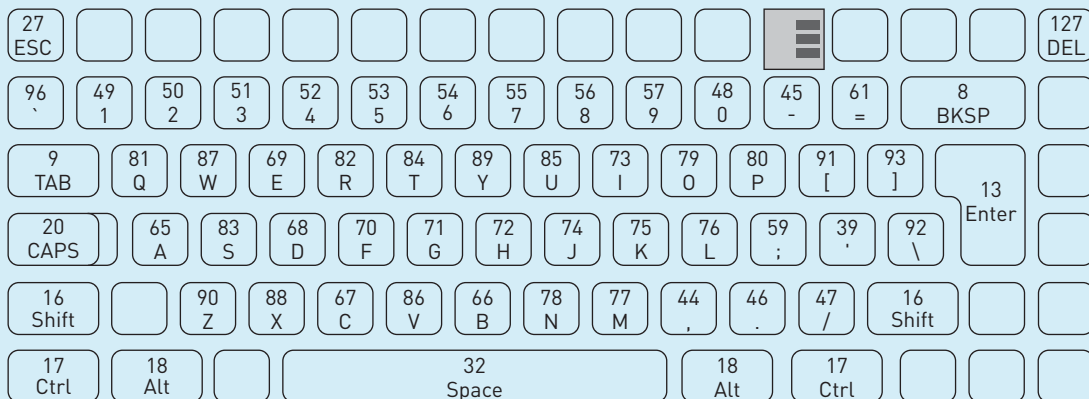


Figure 1.1.13 ASCII codes



1.1.4 ASCII table

- Research design ideas for spaceship characters that will appeal to the target audience. It may be helpful to survey students about the types of characters they like and use this data as inspiration for the new spaceship characters in the ASCII Aliens game.
- Establish rules of the game:
 - The aliens represent a series of 0s and 1s that is the secret code emitted at the start of the game.
 - Players must have an ASCII table with them to decode the secret code.
 - The aliens can only be destroyed by being hit by the BIT they represent: -1 point for each bullet used; -2 points if aliens get to next level, and they move faster to increase difficulty; +5 points for a hit.

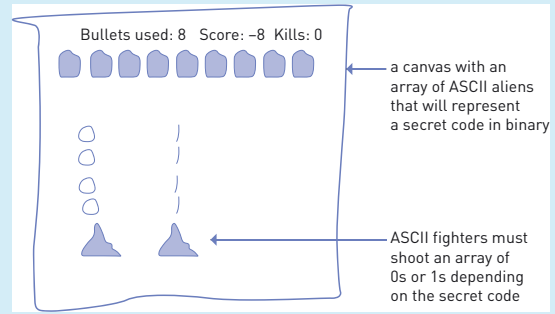


Figure 1.1.14 Design ideas for ASCII Aliens game

3 Decompose the problem into smaller parts.

- Research existing Space Invader games to help break the problem down.
- Create a mind map that breaks the problem down into components.

Figure 1.1.16 shows a mind map decomposing the game into three branches: functionality, user and data.

- *Functionality*: The processes of the game, which will eventually be developed into algorithms.
- *Data*: The ASCII Aliens game needs to keep track of input and output game components for example, score, hits and more.
- *User or player*: Deals with user concerns, for example the user interface and how to play.

4 Identify requirements.

From the mind map it can be determined that the requirements of the digital solution are:

- a user interface with a display area large enough to display several ASCII Aliens and one binary shooter.
- the ability to display an alphanumeric character at the start of the game to establish the correct binary sequence (matching the ASCII code for the character displayed)

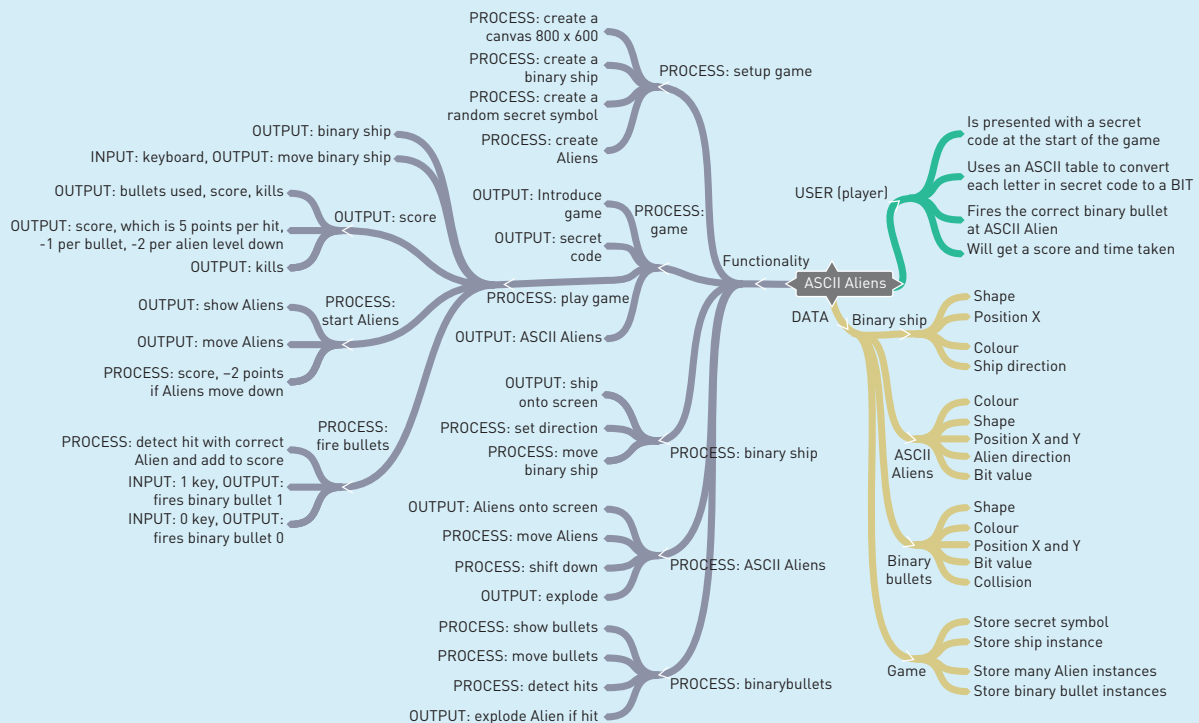


Figure 1.1.15 ASCII aliens mind map

- the ability to capture user keyboard input to control the left-to-right movement of the binary ship and shooting action
- a binary ship that looks appealing to the user and behaves like a space invader's gun
- ASCII Aliens that explode when the correct sequence of 0s and 1s are absorbed
- binary bullets that look like a 1 or a 0 and appear when the user presses 1 or 0. Ideally, bullets should disappear when they collide with an alien.

5 Recognise constraints:

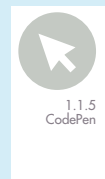
- Coding language and platform may be limited to what is available at school.
- Size of output device will determine the size of the pixel map that can be used for game graphics – this could limit the range of suitable designs for the binary ship and other graphics.

When serious thought and planning goes into the initial exploration, it makes the development of algorithms, ideas and prototypes much easier.

Selecting an IDE

Researching suitable programming environments can also occur during the explore phase of the problem-solving process. Due largely to the availability of debugging tools and cross-platform benefits, Visual Studio Code (VSC) would be an appropriate integrated development environment (IDE) for generating a digital solution for the ASCII Aliens problem.

To generate the digital assets for this case study, VSC was configured for a JavaScript environment and the JavaScript framework p5.js was used to simplify the coding when generating the solution. Being a web-based game, HTML is also required to display the game and link the necessary files such as the p5.js library, which can be downloaded to a local folder or linked to a hosted version. Extensions called Live Server, Debugger for Chrome p5canvas, npm and p5.js snippets can also be useful. This set-up allows viewing of the canvas from within VSC while editing and debugging the code.



INTELLIGENT SYSTEMS: MULTIPLAYER GAME

Your client has requested the development of a physical prototype wireless multiplayer game for children, a demonstration and then a pitch to potential investors with a video 3–4 minutes long. The game must have challenges, both against other players and against a timer. Players receive scores and must be able to interact with the system in some way. Interfaces will need to be created to let players know when it is their turn, whether they succeed or fail, and their score.

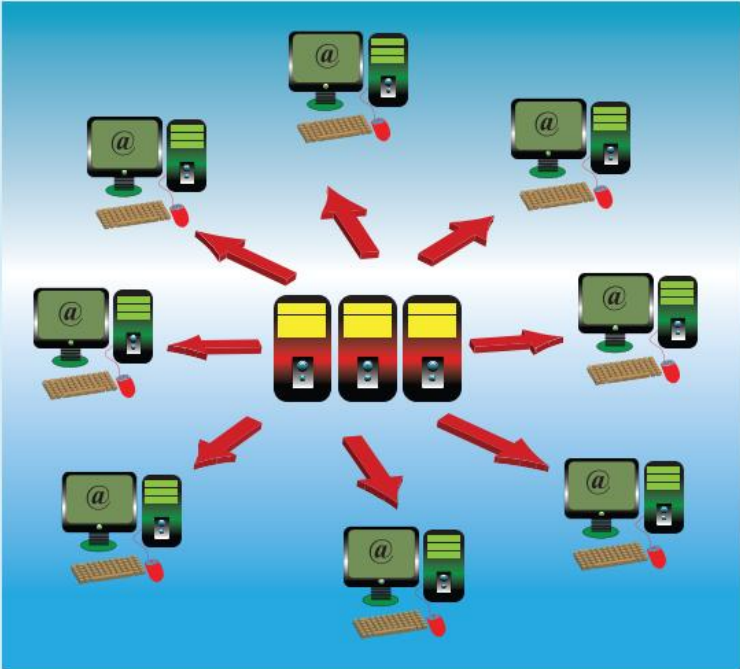
Explore the problem

1 Understand the constituents of a digital problem by restating and summarising.

- Wireless multiplayer game
- Physical prototype
- Interactive
- Compete against other players
- Timer-based and scores

2 Analyse the human need or want.

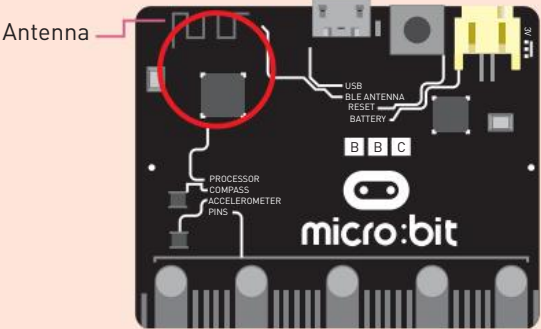
- To satisfy the scenario requirements, this game theme has been inspired by the classic 'bomb game', in which players have a certain amount of time to answer a trivia question before the bomb explodes or they pass to another player without gaining any points. There are 15 questions in total, on cards that lie face down in the middle of the table, next to a 'microbomb'. This appeals to the age group and creates suspense, competition and challenge.
- BBC Microbit hardware can be configured in a one-to-many configuration, as shown in figure 1.1.17. A prototype involving three players and a central microbit to act as a scorer and timer could be used to bring the game to life. To ensure the microbit is capable of these tasks, a hardware input, processing and output (IPO) table is used to *explore* the capabilities.



Depositphotos/oxlock

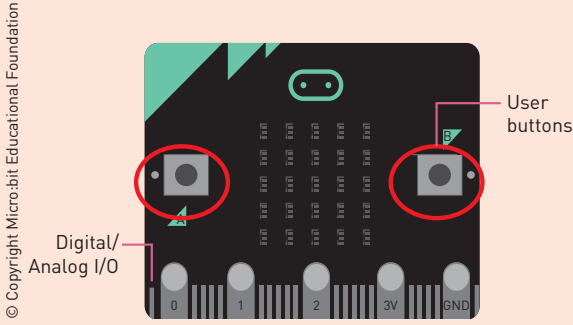
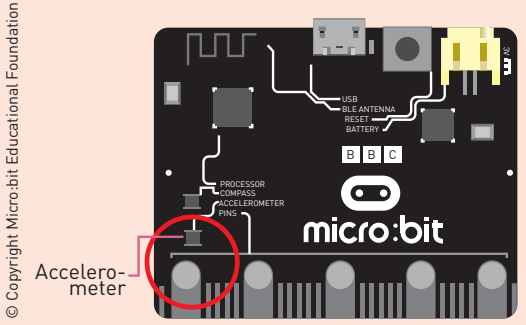
Figure 1.1.16 One-to-many communication

Input (data)	Processing (hardware)	Output (action)
Microbit receiving a radio signal. The input will be wireless radio signals received from another microbit nearby. A key principle of this component is privacy, which is built in from the ground up. So, when you send any data, there is nothing inherent in this protocol that can be used to identify you or your microbit. All devices look identical. Therefore, if you want to be able to identify yourself, you need to add this to your own data.	The central processor unit (CPU) on the microbit is a Nordic Semiconductor nRF51822, which contains a built-in 2.4GHz radio module shown in figure 1.1.18a. For this game, each microbit radio will be placed into a simple mode of operation that allows direct microbit-to-microbit communication using a broadcast radio channel. A (server) microbit will control signals between (clients) player microbits. It provides a channel rate of 1Mbps and a maximum transfer unit of 32 bytes, but reconfigurable up to 1024 bytes. All devices share the same address to guarantee user privacy.	Microbit sending radio signals to other microbits on the same channel. See figure 1.1.18a as this is both input and output.



© Copyright Micro:bit Educational Foundation

Figure 1.1.17a Radio signal

Input (data)	Processing (hardware)	Output (action)
Input buttons A and B on the microbit. One approach that could be considered is that player microbits (clients) use the front A and B buttons as input to allow them to answer the question or pass to another player, as shown in figure 1.1.18b. These buttons will be programmed in the game to send a signal to the server microbit via radio wave from the player.  © Copyright Micro:bit Educational Foundation Figure 1.1.17b Input buttons	A and B are debounced by software, which also includes short press, long press, and 'both A+B' press detection. Both A and B buttons are connected to GPIO pins that are also accessible on the microbit edge connector, however this game will be operated in a wireless mode. They are two tactile user buttons and are debounced using software at a speed of 54 ms period. This means players have time to press the buttons a number of times to choose an answer. The score is also updated when players use button B to answer.	Pull up output signals of A and B buttons operate in a typical inverted electrical mode, where a pullup resistor ensures a logical '1' when the button is released. A logical '0' when the button is pressed. Using this, it can be programmed so that the output might be: Button B pressed once is answer a, pressed twice is answer b, pressed three times is answer c and four presses is answer d. Pressing A passes to another player.
Shaking the microbit. A possible approach could be that the players (clients) shake microbits to check their score and the game microbit (server) shakes to start the game, which resets player scores.  © Copyright Micro:bit Educational Foundation Figure 1.1.17c Accelerometer	Accelerometer in figure 1.1.18c is a separate chip that provides three-axis sensing. In this game all three axes are detected for shaking gesture movement via software algorithms. It is connected to the application processor via the I2C bus. The accelerometer processes movement using a Freescale MMA8653FC with 3 axes, 2/4/8g ranges and onboard gestures. Other gestures are implemented by software algorithms in the runtime.	We could use the accelerometer so that if a player shakes it a wireless signal is sent to the server microbit to send their score. If the server microbit is shaken it will output a signal to reset the game.

Input (data)	Processing (hardware)	Output (action)
A timer. The microbit measures time in milliseconds (1 s = 1000 ms). The microbit has no real-time clock but can measure time elapsed since the timer was reset.	Server microbit acts as a central timer by algorithmically generating a random time interval to use for the duration of the turn.	A random time length at which time a sound will be played.
Audio. A possible input for audio will be a shake of the server microbit that will signal the start to the game. Another input for audio will be from the time when time elapsed has been satisfied.	A possible approach here would be for the server microbit to have the buzzer attached to it via wires. The microbit would also keep track of time taken and output a tone.	'Tick tock' sound to indicate the game is in progress and an alarm when time is up.

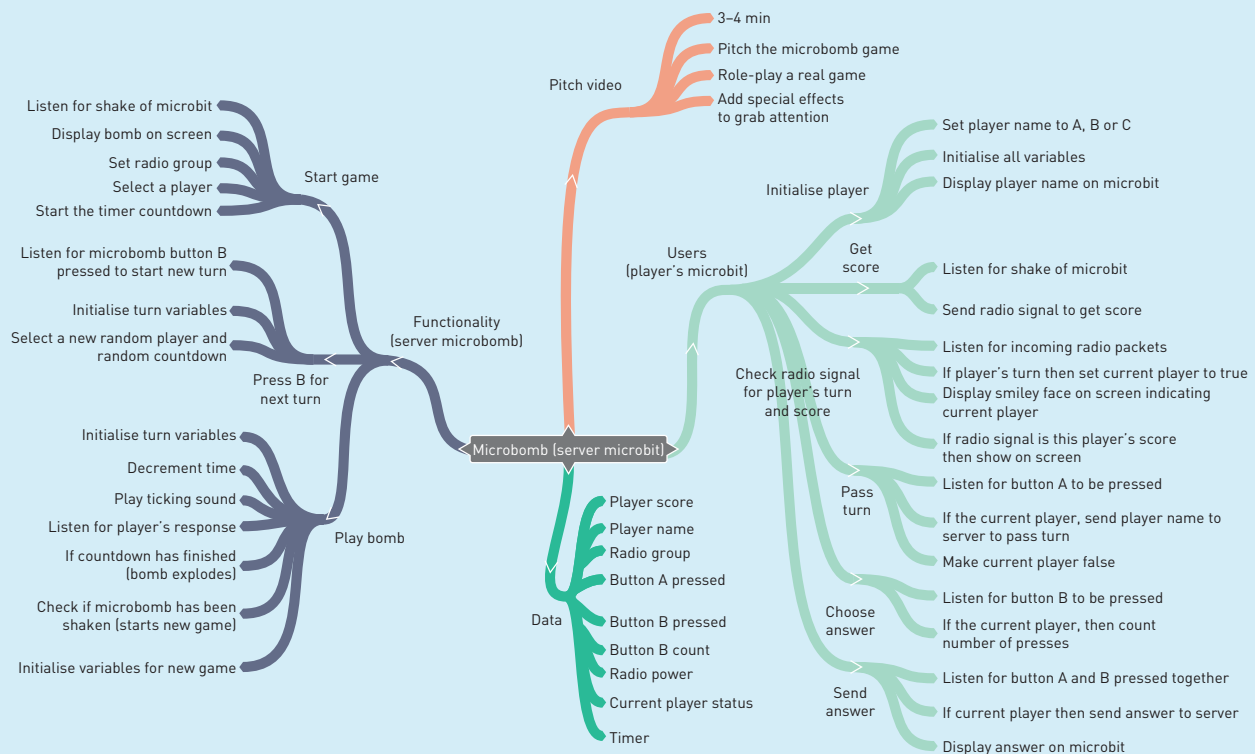


Figure 1.1.18 Mind map of microbomb game

3 Decompose the problem into smaller parts.

- The mind map in figure 1.1.19 breaks the problem into two different programs, one for the central microbomb (server) and the other for the player's microbit (clients). It also highlights the data that travels wirelessly between microbits. The pitch video is added as part of the exploration of the digital problem.

4 Identify requirements.

From the mind map it is possible to identify the following requirements:

- hardware configuration of one microbomb server and three microbit clients
- software generation of server software to control the microbomb server
- software generation of client software to control the player microbits and respond to button presses
- presentation of a pitch to demonstrate how the problem-solving process has been used to generate the prototype solution and demonstrate the game play.

5 Recognise constraints.

The constraints for this solution include:

- Access to hardware components may be limited to specific times during the day at school, limiting the time available for development and testing of the solution.
- Transmission of radio signals between microbits may be limited by distance or subject to interference from other radio sources.

Many of the mind maps and tables presented in the case studies in this section have been developed over a period of time and have been refined several times throughout the problem-solving process. Mind maps, systems thinking, computational thinking and design thinking are all important strategies when exploring digital problems.

TEST YOUR KNOWLEDGE

Exploring digital problems

- 1 Name the four phases in the Digital Solutions problem-solving process.
- 2 An iterative process repeats, and steps in a non-linear process can occur out of order. Describe how the different phases of the problem-solving process can be applied to a digital problem with an approach that is both non-linear and iterative.
- 3 Explain how abstraction is used when exploring a digital problem.
- 4 List the five key steps developers use when exploring a digital problem.
- 5 Which phase of the problem-solving process focuses on ensuring that a digital solution meets the essential requirements?
- 6 Use systems thinking to identify the basic components of a digital solution.
- 7 A mind map can be used to break down a digital problem and brainstorm new ideas. At what phase of the problem-solving process does this occur?
- 8 One of the first steps in exploring a digital problem is to break the problem down. What advantages are there in decomposing a digital problem into smaller subsystems?
- 9 Create an IPO chart from the mind map that resulted from exploring the shop owner's sales alert in figure 1.1.4.
- 10 Explain systems thinking and how it is used in understanding digital problems.
- 11 Explain computational thinking and name the six computational thinking strategies.
- 12 Explain how using logic in digital solutions can be helpful to generate a programmed solution.
- 13 Define decomposition.
- 14 Explain algorithmic thinking and give an example of it.
- 15 How does computational thinking differ from systems thinking?
- 16 Is design thinking important when exploring digital problems? Explain your answer.
- 17 Explain the term 'divergent thinking' in terms of design thinking.
- 18 How does an IPO chart differ from a mind map?



APPLY YOUR KNOWLEDGE

- 1 You have been asked to use the first stage of the problem-solving process and explore a solution to a simple web-based calculator for a young child. A simple calculator will only add, subtract, divide and multiply. Since the target audience is young children learning to use a calculator, include some features that will engage them to make it a fun learning experience.
- 2 A travel agent specialising in holidays for 18- to 24-year-olds needs a responsive website that will allow users to use their phone to book holidays and read specials, and that includes a forum to post holiday experiences and photos and a catchy home page to engage the audience. Explore the travel agent's request using the first stage of the problem-solving process.
- 3 Logic is an important strategy in computational thinking. Explore the following problem:
 - There are four people: Roberta, Thelma, Steve and Pete.
 - Among them, they hold eight different jobs.
 - Each holds exactly two jobs.
 - The jobs are chef, guard, nurse, clerk, police officer (gender not implied), teacher, actor and boxer.
 - The job of nurse is held by a male.
 - The clerk is the chef's husband.
 - Roberta is not a boxer.
 - Pete has no education past Year 9.
 - Roberta, the chef and the police officer went golfing together.Question: Who holds which jobs?

1.2

USER EXPERIENCES AND INTERFACES

Getty Images/BSIP/UiG



Figure 1.2.1 A French-built Nao robot

In this section, students explore user experience and interfaces. Students use design thinking to apply useability principles such as accessibility to worked examples. Students also explore the other useability principles: effectiveness, safety, utility and learnability. Students evaluate existing user interfaces with useability criteria and determine how user characteristics influence the user-interface requirements. Students communicate ideas and technical details through site maps and wireframes, and make recommendations for improvements to user experiences.

User interfaces

Every digital solution requires a user interface that enables interaction in some way. When considering user experiences and interfaces,

the traditional assumption that a ‘user’ is a human being is not as straightforward as it once was. Intelligent systems – such as robots, like the one in figure 1.2.1, and drones – quite happily interact with each other and so become users in this environment. A user experience can be considered to be those aspects of digital solutions that affect how an end user interacts with a digital system. Visual design, interface and navigation design, user needs, and functional requirements need to be considered when developing ideas about user interfaces. User experience design also considers factors that engage the emotions of the user to create a positive experience. Users should not be encumbered by the interface to a digital solution and should be able to predict what will happen in response to each of their actions. Creating an environment in which tasks are carried out almost effortlessly requires a great deal of exploring and design thinking. Good interface design will also minimise input errors. There are a number of principles that can be applied to accomplish successful user experiences based on the useability principles of accessibility, effectiveness, safety, utility and learnability.

Useability principles

An interface is considered to have good useability if it is useful, learnable, accessible, effective and safe.

Utility

The utility of an interface is simply how useful it is. Interfaces must provide appropriate functions to allow users or connected devices to provide input and access output. Factors that improve the operability of the interface, such as portability and responsiveness, contribute to its utility.

Portability

An interface is said to be portable if it is platform-independent. This means the application does not rely on hardware or a particular operating system to run. The methods for achieving portability and the extent to which portability can be applied depend largely on the technology context. Use of open-source, rather than proprietary, systems can improve interface portability in intelligent systems. Testing user interfaces on multiple platforms can help improve the portability of interactive media and mobile applications. Ensuring a web page displays appropriately in multiple browsers improves the portability of web applications.

Responsiveness

A responsive design will dynamically adjust to suit the screen size and capabilities of a wide variety of devices. For example, a mobile phone has a lot less screen area than a PC, so thought needs to go into what essential parts of the interface should always be visible and what should be hidden behind icons and menus on a smaller layout. Mobile and PC monitor screens do not all have the same dimensions, so generating the user interface with code that uses variable data for screen height and width is an important part of creating a responsive display for interactive media and mobile applications. Some screens allow touch and others do not, but responsive solutions respond appropriately to the capabilities of the device. As shown in figure 1.2.2, a responsive web application interface will have a very different layout on different screen sizes. An example of code used to generate a responsive user interface for a web application is provided in section 1.4.

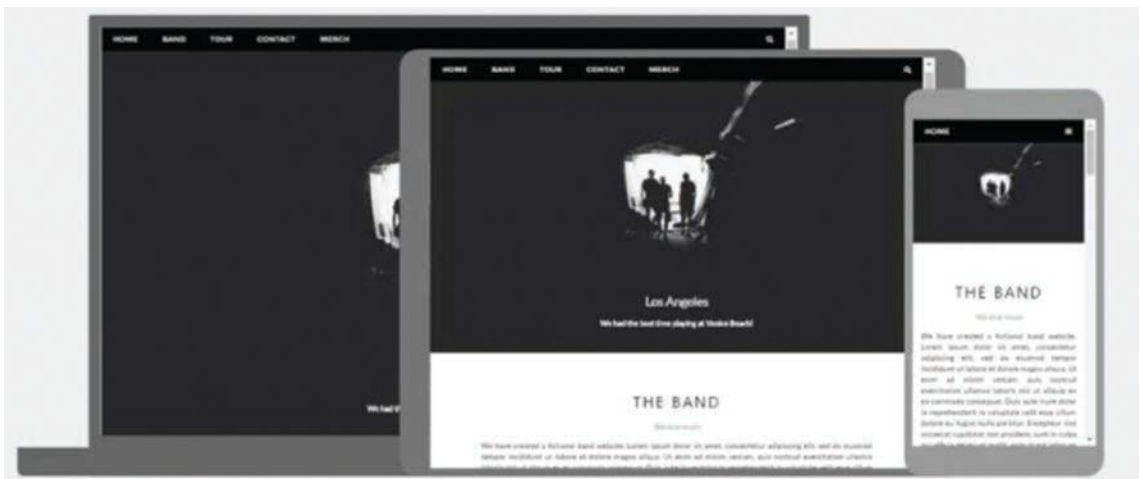


Figure 1.2.2 Responsive design

In responsive web application layouts, the viewport is the area of a web page that is visible to the user. It varies with the device, and will be smaller on a mobile phone than on a computer screen. Some websites change style more than once depending on the viewport size of the browser. Often, the design will change at tablet size and then again at mobile phone size, as shown in figure 1.2.2. In the context of web design, these 'change in style' points are known as breakpoints.



ACTIVITY 1.2.1

In groups, use your browser to explore the Internet for examples of responsive pages. You can test for responsiveness by shrinking the browser down below the viewport size. Develop a list of sites with multiple breakpoints and document the viewport size at each breakpoint. This is done by taking a screenshot when the browser window seems about right, and measuring the width of the browser in pixels. If possible, right-click to view the source HTML and attempt to identify the viewport meta tag. Typically, it will appear at the top of the page and look something like this: `<meta name="viewport" content="width=device-width, initial-scale=1.0">`

Learnability and effectiveness

Effectiveness of a user interface relates to how well it does what it needs to do. Users need to be able to interact with a digital solution quickly and easily, and outputs need to be presented clearly in a way that can be easily interpreted by the user. An effective user interface allows users to complete their interactions accurately and efficiently without requiring a detailed understanding of the underlying complexity of the system.

It takes less time and user effort to learn how to use a digital solution effectively if an interface is learnable. A very learnable interface is said to be intuitive, because the user can immediately grasp how to interact with it. Learnability includes how quickly and easily users can learn to find information, complete tasks and navigate the digital solution. The design of a user interface, in both layout and content, is critical for learnability, and CARP offers a set of aesthetic visual design principles that will improve the learnability and effectiveness of a user interface.

CARP design principles

CARP stands for **contrast**, **alignment**, **repetition** and **proximity**.

- **Contrast** insists that any item you would like to stand out and be eye-catching should contrast with its background and use other effects such as bold type, underlining and more. It is important, however, not to overuse contrast; otherwise you risk losing your message completely.
- **Alignment** is about positioning graphical components. Alignment occurs on an imaginary line, either horizontal or vertical. By aligning certain elements, the designer can create visual chunking of content that eases reading and helps the user see connections in the material. One type of alignment method popular in visual design is the rule of thirds.
- **Repetition** of elements, such as fonts, colours and shapes, creates a uniformity that promotes design harmony. Consistent use of a single font or colour can convey a feeling or emotion.
- **Proximity** requires that all related information should be organised in the same area on the page, rather than scattered about and making it difficult for users to understand the page.



Figure 1.2.3 CARP design principles in use in a web page: (a) without CARP; (b) with contrast; (c) with alignment and repetition; (d) with proximity

ACTIVITY 1.2.2

- 1 Identify where and how CARP has been applied in figure 1.2.3. Discuss with your group.
- 2 How could CARP be applied to the navigation design of an app?
- 3 Read the useability success criteria in table 1.2.1 and identify which tips relate mostly to effectiveness, which relate mostly to the learnability of a user interface and which contribute to both effectiveness and learnability.
- 4 Choose the five most important tips and create your own learnability and effectiveness checklist.
- 5 Use your checklist to evaluate a school application.
- 6 Discuss your findings with the class.



1.2.1 CARP infographic

Table 1.2.1 Useability success criteria

VISIBILITY OF SYSTEM STATUS	Inform users of system operations with easy to understand and highly visible status displayed on the screen, e.g. breadcrumbs, loading screen.
MATCH BETWEEN SYSTEM AND THE REAL WORLD	Reflect the language and concepts of the target users and what they would find in the real world. Presenting information in logical order based on user's expectations derived from their real-world experiences will reduce cognitive strain and make systems easier to use.
USER CONTROL AND FREEDOM	Offer users a digital space where backward steps are possible, including undoing and redoing previous actions.
CONSISTENCY AND STANDARDS	Interface designers should ensure that both the graphic elements and terminology are maintained across similar platforms. For example, an icon that represents one category or concept should not represent a different concept when used on a different screen.
ERROR PREVENTION	Whenever possible, design systems so that potential errors are kept to a minimum. Users do not like being called upon to detect and remedy problems, which may on occasion be beyond their level of expertise. Eliminating or flagging actions that may result in errors are two possible means of achieving error prevention.
RECOGNITION RATHER THAN RECALL	Minimise cognitive load by maintaining task-relevant information within the display while users explore the interface. Human attention is limited and we are only capable of maintaining around five items in our short-term memory at one time. Due to the limitations of short-term memory, designers should ensure users can simply employ recognition instead of recalling information across parts of the dialogue. Recognising something is always easier than recall because recognition involves perceiving cues that help us reach into our vast memory and allowing relevant information to surface. For example, we often find the format of multiple choice questions easier than short answer questions on a test because it only requires us to recognise the answer rather than recall it from our memory.
FLEXIBILITY AND EFFICIENCY OF USE	With increased use comes the demand for fewer interactions that allow faster navigation. This can be achieved by using abbreviations, function keys, hidden commands and macro facilities. Users should be able to customise or tailor the interface to suit their needs so that frequent actions can be achieved through more convenient means.
AESTHETIC AND MINIMALIST DESIGN	Keep clutter to a minimum. All unnecessary information competes for the user's limited attention and could inhibit memory retrieval of relevant information. Therefore, the display must be reduced to only the necessary components for the current tasks, while providing clearly visible and unambiguous means of navigating to other content.
HELP USERS RECOGNISE, DIAGNOSE AND RECOVER FROM ERRORS	Designers should assume users are unable to understand technical terminology, therefore error messages should almost always be expressed in plain language to ensure nothing gets lost in translation.
HELP AND DOCUMENTATION	Ideally, we want users to navigate the system without having to resort to documentation. However, depending on the type of solution, documentation may be necessary. When users require help, ensure it is easy to find, specific to the task at hand and worded in a way that will guide them through the necessary steps towards a solution to the issue they are facing.

Source: from 'User Interface Design Guidelines: 10 Rules of Thumb', by Euphemia Wong, Interaction Design Foundation CC BY-ND 4.0

Accessibility

When we develop ideas for a digital solution, we should always consider how accessible the final product is for all users. Web applications should be developed using conventions set by the World Wide Web Consortium (W3C) so all Internet users have equitable access to information. It is now mandatory in many countries to develop applications that are able to be read by screen readers. When you explore a digital problem, you should consider:

- which groups of people need to be considered, and why
- which tools different people use to interact with your digital solution
- how to make accessibility part of the development process.



1.2.2
Accessibility

In the web context, for example, we know that screen readers can be used to read the content of a HTML page. However, this is only possible if the developer has followed a set of accessibility conventions known as the Web Content Accessibility Guidelines (WCAG). Some excellent accessibility resources and tutorials are available through the Moz://a website.



ACTIVITY 1.2.3

Split into five groups. In your group, go to the Accessibility page on the Moz://a website and explore *one* of the topics below. Present a summary of your findings to the class.

- HTML
- CSS and JavaScript
- WAI-ARIA
- Multimedia
- Mobile



Figure 1.2.4 Poorly designed web page

Figure 1.2.4 is an example of a badly designed web page, created as part of the Rusty and the Ezibeats project introduced in section 1.1. We can use this badly designed user interface to explore a number of accessibility issues. Although many of the examples we are about to explore are for the web application technology context, the same principles also apply to other technology contexts. Assistive technologies such as eye trackers, screen readers and voice recognition systems work better if developers follow the accessibility conventions for their specific digital technology context.

Text colour contrast

The first accessibility issue is text colour contrast, which you can explore further at the Vision Australia website. Here you can download a colour contrast analyser that allows you to measure the contrast between any two colours. Although initially developed for web application accessibility, it can be used to identify contrast issues with a range of user interfaces.

There are many online tools that allow you to analyse text colour contrast problems on a web page. Google's colour contrast analyser extension for the Chrome browser can be used to analyse text over the top of background images or gradients, analyse text in images or get a visual overview of the contrast between elements of a web page. The Chrome extension was used on figure 1.2.4, with the result in figure 1.2.5. You cannot see the menu text or body text. This test is important when you are exploring and developing interface ideas for web applications.

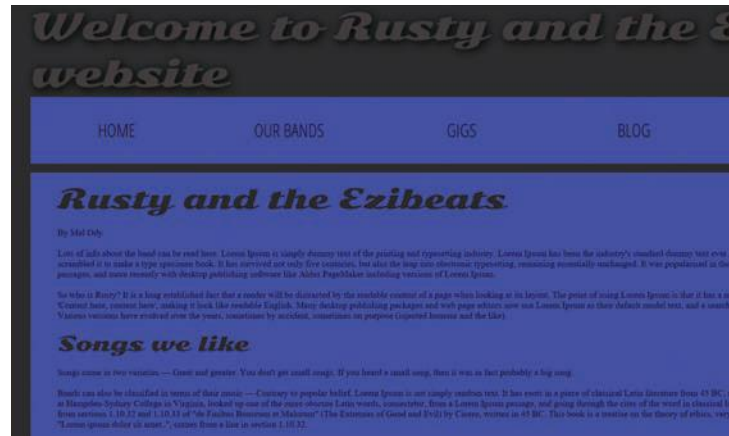


Figure 1.2.5 Result when Chrome colour contrast analyser is applied

Screen reading

The second accessibility issue with the badly designed page is the screen reading. To explore screen reading and how it works, you can again go to the Vision Australia website.

To run a screen reading test on the navigation menu of the badly designed Rusty and the Ezibeats web page, the developer used Google Chrome's extension ChromeVox. In figure 1.2.4, the navigation options are 'Home', 'Our bands', 'Gigs' and 'Blog'. When the mouse is moved over these options with ChromeVox extension installed, it does not recognise that this is a navigation bar. This is because semantic HTML5 tags are not being used. The wrapping HTML is:

```
<div class="nav">
Menu content in between
</div>
```

The tags above still work, but the screen reader does not detect that it is a navigation bar. The tag '<div>' has been extensively used in the past to create a division on a web page. The latest version of the HTML standard (HTML5) introduced **semantic tags**. These are tags designed to identify a specific part of a web page. Using them assists people who rely on screen readers, and makes your website more accessible. For example:

```
<nav>
Menu content in between
</nav>
```

Accessible images

The third accessibility issue with the page is with images, such as that shown in figure 1.2.6.

At present, the screen reader does not describe the images when the page is read. This is because the HTML (below) is missing the '<alt>' attribute, or alternative text.

ACTIVITY 1.2.4

Explore WCAG 2.0 and summarise its purpose. Try running the text contrast test on a web page you suspect might not demonstrate good contrast, and summarise your results.



1.2.3 Web design and applications: accessibility

ACTIVITY 1.2.5

Install a screen reader and test some web pages. You can view the source HTML by right-clicking and then clicking 'View page source'. Observe whether semantic HTML has been used. You can explore the semantic HTML tags at the [Moz://a](https://moz.com/a) website.



1.2.4 Screen reader



Getty Images/Rolf Brenner

Figure 1.2.6 Image on a web page



```

```

Alternative text is used for several reasons, one of which is to be detected by screen readers. All that is needed is:

```

```

Alternative text can also be provided for images in other types of technology contexts.

Accessible multimedia

The fourth accessibility issue is an audio file that is not available to users who are hearing-impaired. Currently the HTML for the audio is:

```
<audio controls>
  <source src="media/audio.mp3" type="audio/mp3">
  <source src="media/audio.ogg" type="audio/ogg">
  <p>It looks like your browser doesn't support HTML5 audio players.</p>
</audio>
```

Audio transcripts should be made available to ensure accessibility of your audio files. These can either be included on the same page as the audio in some way, or included on a separate page and linked to. There are a number of ways, both paid and free, to transcribe an audio file. YouTube is a freely available medium that transcribes your file automatically when you upload it. If you use an automated service, then you will probably have to use the user interface that the tool provides.

Safety

Your interface design must also consider measures that help create a safe environment for the user:

- *consent prompts* – when a user is prompted with a choice to give consent when an app seeks to use sensitive data. Some prompts are automatically built into the device operating system or web browser and appear when an app accesses a phone sensor such as the geolocation sensor, as illustrated in figure 1.2.7.
- *icons* – for user safety; for example, the padlock icon to indicate that data is secure (encrypted) or the exclamation mark for an alert
- *hiding data* – protection from prying eyes; for example, when a password is to be entered it should be hidden using asterisks or dots
- *data loss* – if a user attempts to delete files unintentionally, a warning might appear, such as that in figure 1.2.8.

The Apple Developer website provides guidelines for developing a secure user interface.

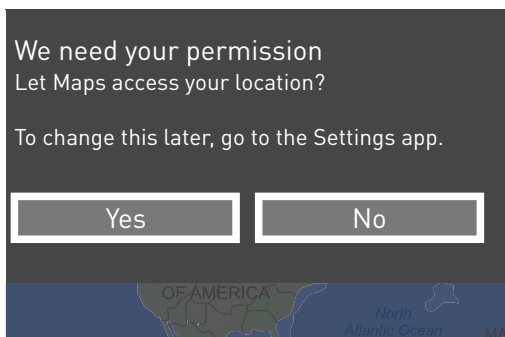


Figure 1.2.7 User interface privacy protection

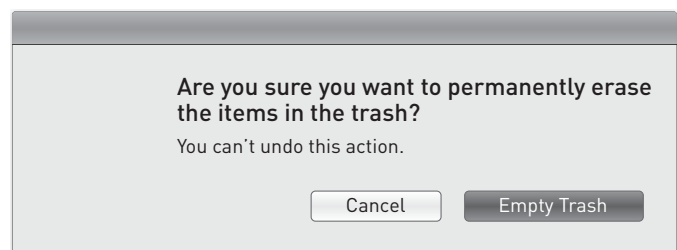


Figure 1.2.8 Building in user prompts for safety

CASE STUDIES

...

WEB APPLICATION: USER INTERFACES

In section 1.1, using the Digital Solutions problem-solving process, we explored and developed ideas for a prototype responsive website for Rusty and the Ezibeats. Here we focus further on the development of the user interfaces by sketching a wireframe layout of each page. Figure 1.1.12 is the mind map from our exploration of ideas. Based on that, six pages needed to be developed: Home, Band, Tour dates including ticket purchase, Contact information, Merchandise, Forum and sign-in for forum. While analysing the human need or want, or afterwards, ideas for the user interface should be sketched and existing solutions should be explored and evaluated.

Wireframes are the next step in the technical development of the user interface. Wireframes visualise the layout and can include technical information, such as dimensions, navigation, functionality and suggested styling, that will be used in the next phase of the Digital Solutions problem-solving process to generate the user interfaces. Two sample pages have been chosen to illustrate wireframing: the home page and Tour/Purchase page.

Home page wireframes are intentionally kept simple and engaging.

Table 1.2.2 Home page responsive wireframe concepts

	Canvas 1024 × 600 NavBar 1024 × 40 Rollover buttons 100 × 40 – black with white font, rollover grey with black font Dropdown button for access to Merchandise and Forum Search button to search site Rotating banner 1024 × 560 Styling White font Black background
	Canvas 320 × 480 NavBar 320 × 40 Buttons 75 × 40 Hamburger dropdown button for access to Band, Contact, Merchandise, Tour dates Rotating banner 320 × 200 Styling White font Black background



Table 1.2.3 Tours and ticket booking wireframes

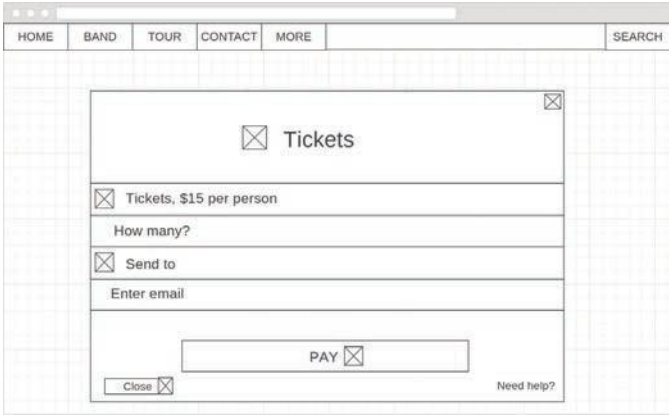
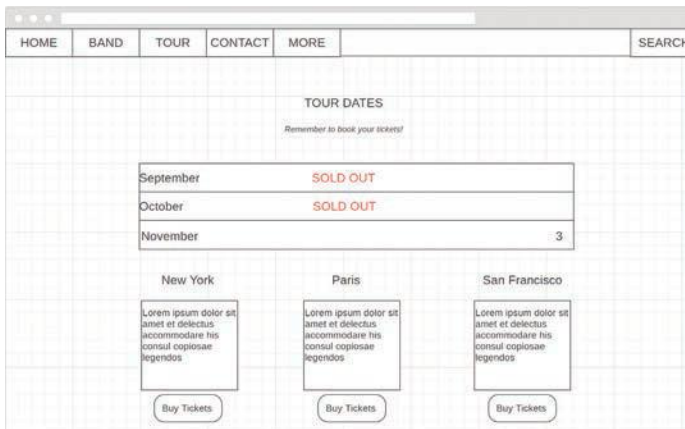
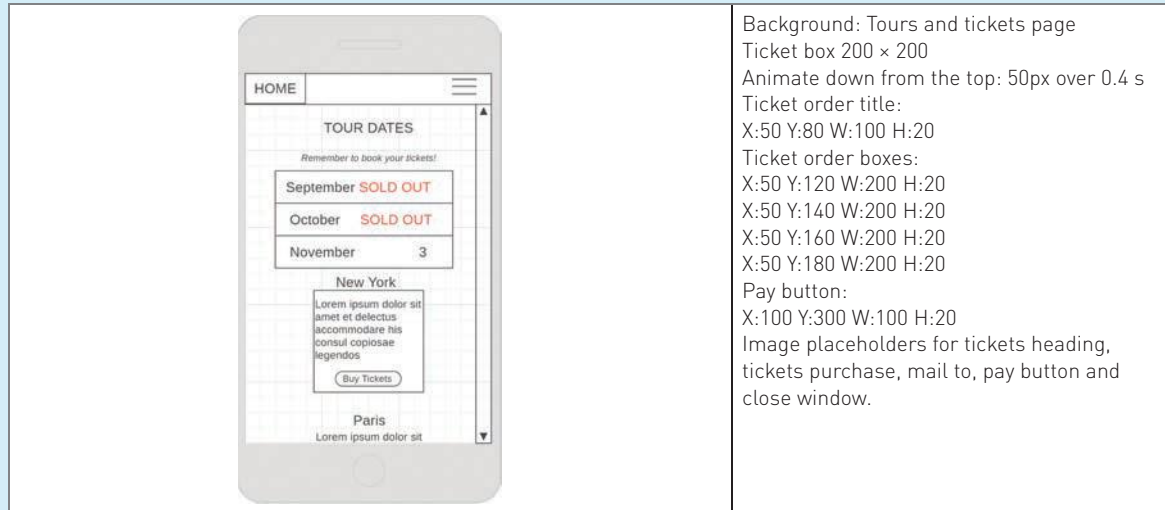
	Nav bar and style same as above Ticket order form: Title box X:400 Y:100 W:200 H:60 Ticket listbox: X:200 Y:200 W:640 H:40 Ticket order titles: X:230 Y:360 W:100 H:20 X:460 Y:360 W:100 H:20 X:705 Y:360 W:120 H:20 Ticket order boxes: X:200 Y:400 W:140 H:130 X:440 Y:400 W:140 H:130 X:680 Y:400 W:140 H:130 Buttons: X:220 Y:540 W:100 H:40 X:460 Y:540 W:100 H:40 X:700 Y:540 W:100 H:40
	Canvas 320 × 480 NavBar 320 × 40 Buttons 75 × 40 Hamburger drop-down button for access to Band, Contact, Merchandise, Tour dates Scrollbar X:300 Y:40 W:20 H: 440 Title box X:60 Y:60 W:200 H:60 Ticket listbox: X:40 Y:210 W:230 H:40 Ticket order titles: X:120 Y:260 W:100 H:20 X:120 Y:440 W:100 H:20 X:120 Y:620 W:120 H:20 Ticket order boxes: X:90 Y:280 W:140 H:130 X:90 Y:460 W:140 H:130 X:90 Y:640 W:140 H:130 Buttons: X:120 Y:385 W:85 H:20 X:120 Y:460 W:85 H:20 X:120 Y:680 W:85 H:20

Table 1.2.4 Wireframes when 'purchase ticket' button is pressed

	Background : Tours and tickets page Animation: The tickets form slides down from the top and a fade-in transition. Tickets box: 800 × 400 Animate down from the top: 300px over 0.4 s Ticket order title: X:230 Y:360 W:100 H:20 Ticket order boxes: X:200 Y:400 W:800 H:20 X:200 Y:420 W:800 H:20 X:200 Y:440 W:800 H:20 X:200 Y:460 W:800 H:20 Pay button: X:400 Y:500 W:400 H:20 Image placeholders for tickets heading, tickets purchase, mail to, pay button and close window.



You can also add padding, margins, font types, animation effects such as rollover or fading in and out, and any other detail that might be part of your technical planning. Understanding the user persona will also have an effect on your design choices.

ASCII ALIENS INTERACTIVE MEDIA (LEARNING OBJECT): USER INTERFACES

In section 1.1 we explored the game Space Invaders to reinforce how the ASCII table is used to convert characters (human language) to binary (machine language) using a simple, fun, interactive approach that applies game theory to engage the user. The following list is a non-exhaustive guide to user interface and functionality considerations to explore when generating games or learning objects. They are also measurable and can therefore be used to develop evaluation feedback checklists:

Attractiveness, enjoyability

- ☐ Explore and develop the entertainment goal. Ask yourself, 'What is supposed to make the game/ learning object entertaining?' Ideas such as:
 - ☐ quality of graphics
 - ☐ sound
 - ☐ animation.

Education

- ☐ If it is a learning object, explore and develop the educational goals.
- ☐ Examples such as 'escape the room', simulations, quiz-based

Useability

- ☐ **Ease of learning:** Can the system be learnt quickly and easily?
 - ☐ Clear instructions (not lengthy)
 - ☐ Available cheats/hints (not too easy)
 - ☐ Introduction/conclusion
- ☐ **Ease of use:** Can the game be accomplished rapidly, accurately, and without undue effort?

Staged difficulty

- ☐ Start easy but increase difficulty to stimulate thrill of accomplishment. Games are not fun without difficulty. Consider levels of difficulty or scoring mechanism.
- ☐ What needs to be difficult about it to make it fun?
- ☐ Levels of difficulty can be built in many ways, such as:
 - ☐ scores (common to games)
 - ☐ bonuses (common to games)
 - ☐ timing (common to games)
 - ☐ levels.



Develop



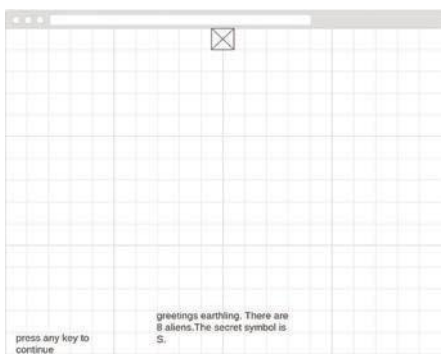
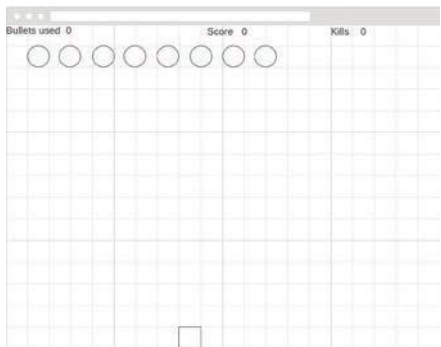
Evaluate



User interfaces for interactive media

In section 1.1, we developed and illustrated ideas for the ASCII Aliens game. We can now start to develop ideas for the user interface by using wireframes to illustrate technical details such as layout and functionality.

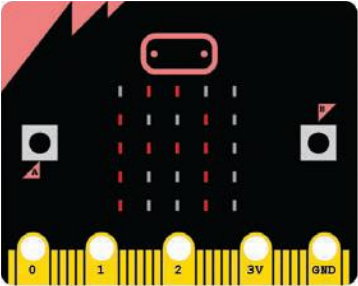
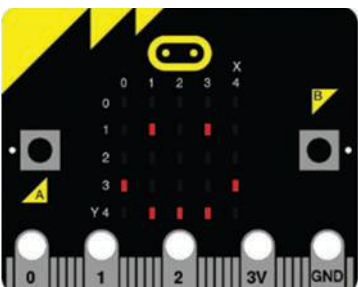
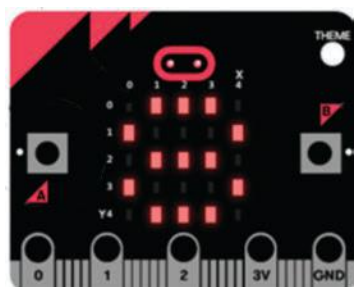
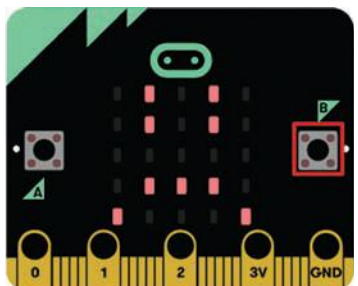
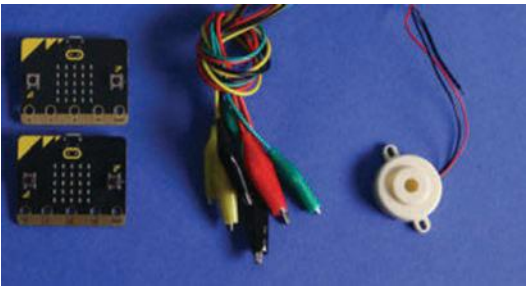
Table 1.2.5 Wireframe for ASCII Aliens in the development of user interfaces

	Function: Intro 1 Background image 800 × 600 stars and space Text font size 32, colour (255,102,153) Title X:300 Y:50 Message1 X:250 Y:70 W:400 H:400 Message2 "press any key" X:250 Y: bottom – 20
	Function: Intro 2 Background image 800 × 600 stars and space Function: loadalien Alien W:50, ship fly in from the left. Stop in middle. Then animate down a beam carrying the secret random symbol. When it reaches the bottom display the secret symbol at X: middle – 70 Y:bottom – 70 Message : "Press any key to continue" X:50 Y: bottom – 20
	Function: playGame Bullets used X:10, Y:50 Display variable ammo Score X:300, Y:50 Display variable score Kills: X:500, Y:50 Display variable kills Move alien array 1 pixel at a time across the screen until it hits the edge then move in opposite direction and move down 20 and move faster by 0.1. If alien is hit then display the binary value and 2 added to score. Binary ships move left and right with arrow keys. Binary bullets are fired with 0 button and 1 button.

Microbomb intelligent systems: user interfaces

In section 1.1 we explored and developed some ideas for a prototype multiplayer game 'Microbomb', which uses a client-server style network architecture and the programmable BBC Microbits. This means a central microbit (server), which is the microbomb, runs the game, while all players have a microbit (client) that communicates to the server. The user interface considerations for this style of game need to consider what the user sees on their microbit and what all players will see on the server microbit. The interface screen of a microbit is a 5 × 5 LED display that is quite basic and limits what can be displayed. An important point here is that a user interface does not have to be a screen displayed to a human. It can be made up of hardware to perform output for a user – in this case, a sound buzzer for humans and a set of radio signals for communicating microbits. The wireframes in this case would be quite basic (see Table 1.2.6).

Table 1.2.6 User interface for an intelligent system

© Copyright Micro-bit Educational Foundation 	When the central server microbot is shaken at the start, display the letter of each player on their Microbit A, B or C and display a bomb image on server microbit. This interface is also shown on the server to display the answer chosen by the player. Button B to start new turn and shake server to start new game.
© Copyright Micro-bit Educational Foundation 	Display a smiley face on the microbit of the user who has been randomly selected by the server to play. Button B used to answer question. Button A used to pass to another player and a smiley face appears on other player to indicate their turn.
© Copyright Micro-bit Educational Foundation 	Display the score on a user's screen if they shake their microbit.
© Copyright Micro-bit Educational Foundation 	Display a sad face on the user screen if the bomb blows.
	Buzzer used to connect to the server microbit to indicate game is counting down and then final explosion.
RADIO GROUP	This sets a radio channel on each microbit to allow them to signal to each other. This is an interface for microbits.



TEST YOUR KNOWLEDGE

- 1 Explain what is meant by responsive design.
- 2 Explain what a breakpoint is in terms of responsive design.
- 3 What does the acronym CARP stand for? Briefly explain each term.
- 4 How could CARP be applied to the navigation design of an app?
- 5 Explore the Internet for a good example of each CARP element, and explain why it is a good example.
- 6 List five key qualities of a learnable user interface.
- 7 Who sets the standards for web accessibility?
- 8 List some possible accessibility issues that might occur on a web page and ways in which they can be prevented.
- 9 What is the purpose of WCAG 2.0?
- 10 Explain how you can create a safe user interface.
- 11 What is a wireframe and what information can be added to it?
- 12 In what situations might unintentional data loss occur?
- 13 List some user interface and functionality considerations to keep in mind when exploring and developing interactive media.
- 14 Give some examples of different types of user interfaces, both human and non-human.

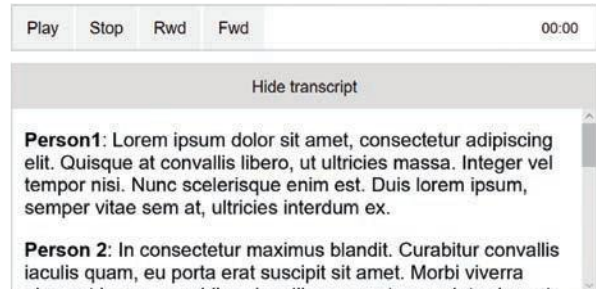


APPLY YOUR KNOWLEDGE

- 1 Responsive design is the practice of making the layouts and other features of apps change dynamically depending on factors such as screen size and resolution, so they are useable and accessible to users of different device types. Apply your knowledge of the responsive design principles learnt in this section to develop wireframes for the travel agent website you explored in 'Apply your knowledge' from section 1.1 to communicate the prototype concepts.
- 2 Identify and evaluate the useability features of a website. Create a criteria checklist to measure the success of its useability features.
- 3 Explore, develop, generate and evaluate your own accessible audio file by recording your voice and then developing the following interface.



When the button 'Show transcript' is clicked, the box will expand, like the following example, but with your transcript.



You can see a solution at the CodePen website, but it is best to have a try first. The solution will not have the audio file uploaded but will demonstrate the transcript and audio player.

1.3

ALGORITHMS AND
PROGRAMMING TECHNIQUES

In this section, students explore creative and technical aspects involved in developing interactive digital solutions. This is done by recognising and describing programming syntax and rules, and understanding that simple algorithms consist of input, process and output. Students develop an understanding of basic algorithm constructs – assignment, sequence, selection, conditions, iteration and modularisation – and how to use them. Students will represent algorithms using pseudocode, symbolise algorithms with diagrams, and understand basic features of programming, good practices and conventions. The purpose of syntax, the scope of local and global variables, and object/event triggers will be identified and described. Students will explore development tools, use a procedural text-based language, and analyse the end result of code statements and re-useable coded components.

Becoming a programmer

Creating with code is a highly relevant, sought-after skill in a modern world surrounded by digital technology. Learning to code empowers you to go beyond using apps and become a creative developer, making digital solutions. To begin programming requires careful consideration, planning, and learning the basics.

Many people have already written code, stored in libraries, which can be used in your digital solution. You need to develop skills allowing you to keep up with changes in versions of languages, understand libraries of code, modify and test code, and insert code into your specific digital solution. The first step in creating with code is to learn how to structure and represent algorithms. This is a way to represent your step-by-step solution that is free of programming code and follows good algorithm design principles.

Programming gives you the power to control a digital device. Section 1.1 explored the need to understand hardware and software subsystems in order to create a digital solution. Let's explore the idea of controlling digital devices.

Controlling digital devices

A computer, a tablet, a smart device, a robot and many other digital devices can all be connected to make up what we know today as the Internet of Things. For example, in figure 1.3.1, streaming video from the vehicle cam is being processed in real time for further processing used in the automation of driving. Every complex digital solution is made up of many smaller hardware components with circuitry, wires and electricity and the capacity to be programmed to do simple tasks, such as performing basic arithmetic, comparing two numbers, loading a value from memory

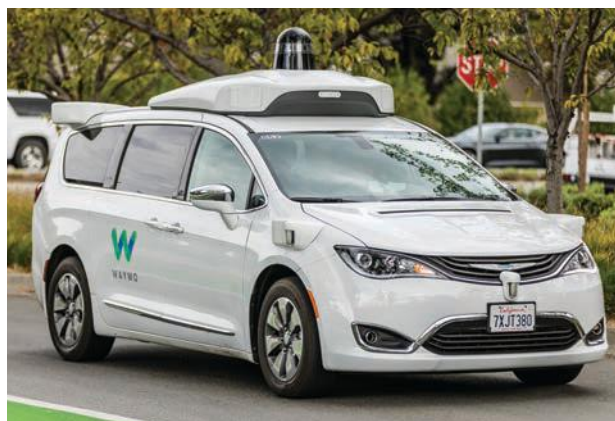


Figure 1.3.1 Google automated driving



1.3.1 Learn to code
1.3.2 Computer programming
1.3.3 Learn to code
1.3.4 Code wars
1.3.5 HTML: the language for building web pages



1.3.6 Text to binary translator

or storing a value to memory. These very basic commands need to be converted to a set of 0s and 1s suitable for electrical signals that computers can process. Binary, which is the number system represented by 0s and 1s, suits this two-state environment of on and off and is known as the lowest level of machine language. All data and commands in a digital solution can be represented using binary states. There is even a binary value for the letter 'A'.

From human language to machine language

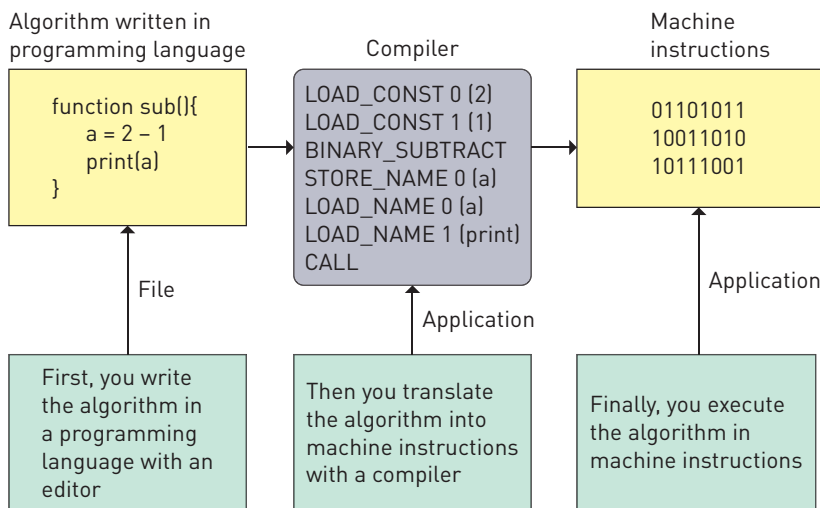


Figure 1.3.2 From human language to machine language

Programmers are ultimately the ones who make a device 'smart', allowing the computer to do all the amazing things we expect. It is important to have a basic understanding of the process from human to machine language. Humans (and, in modern times, not only humans) write code that is then compiled or interpreted into machine instructions (illustrated in figure 1.3.2). Conversion from human language into machine code today is not as clear-cut as it used to be. **Compiling** is a translation process from human to machine instructions. Figure 1.3.2 demonstrates the translation of the subtraction (or '-') operation in the human instructions

(algorithm) stored on a file directly to the 'SUBTRACT' in machine instructions (binary). The compiled machine instructions can also be stored in a file on the device, often referred to as an executable file – such as `pacman.exe`, for example – and run (executed) on that device by a user.

Unlike the compiled process, in the **interpreted** process the native device is fed machine instructions line by line at run time by some other intermediate program, such as a web browser, that translates program syntax to binary instructions each time it is run. This is quite different to the compiled process. The interpreted process requires an intermediate program, whereas the compiled process does not.

Computer languages for humans: high-level languages

Learning zeros and ones to encode data such as the happy face in figure 1.3.3 is not a great way to spend your life. Fortunately, higher level programming languages have been developed to let people express

	■			■	
	■			■	
	■	■	■	■	

```

1 1 1 1 1 1
1 0 1 1 0 1
1 1 1 1 1 1
1 0 1 1 0 1
1 0 0 0 0 1
1 1 1 1 1 1

```

commands to the computer in a way that is easier to comprehend. There will always be debate around what programming language to use for solving a particular digital problem, a little like the debates between the fans of Ford or Holden, or Apple or Microsoft. A programming language is what people use to code, and is simply another form of communication, with a strict set of rules known as **syntax**. A programming language is needed to help us organise and generate ideas that can be translated into machine language. It is analogous to the English language, which uses a set of grammar and spelling rules to help us organise and describe ideas for other people.

Figure 1.3.3 Happy face in binary

Gaining employment in software engineering is increasingly about the range of skills an applicant can offer. A good programmer generally knows a handful of languages.

Traditionally, programming has focused on coding well. Modern programming, however, is more than that: it is about developing something. Programming languages evolve over time, and language familiarity is only a piece of the puzzle. The starting place is developing effective and efficient algorithms, and learning structured programming techniques that require a program to be properly designed before coding begins. It is often the case, when developing algorithmic ideas, that there are many correct solutions; the trick is to learn and practice how to develop efficient, effective and reusable algorithms. These are the algorithms behind successful digital solutions, and those who can create and develop clever algorithms will be successful in a modern digital world.

Program data

Computer programs are written to process data. To understand structured programming, you first need to understand the nature and structure of the data being processed and stored. A variable is used in programming to reference data stored in memory. The term ‘variable’ was first used because the stored data value may vary as the program runs – for example, the score in a game. Sometimes, data does not change, and in this case a constant data type is used – for example, acceleration due to gravity on Earth is 9.8 m/s^2 . Data can be stored in program memory using different data structures. Data is known to have a data type, which is based on the value stored. Common numerical data types include ‘integer’ and ‘real’, while letters are usually ‘character’ or ‘string’ (for whole words). The Boolean data type can have two values, ‘true’ or ‘false’. Programming languages have different keywords to represent many different types of data, but in an algorithm it is only necessary to indicate if the data is numerical, text or Boolean.

Simple data structures contain only one value, for example a variable. Complex data structures can be built from a combination of simple data structures. The most common complex data structures are:

- array or list – made up of a number of data items that all have the same data type
- **object** – a collection of data properties and methods that all have some relationship to the object.

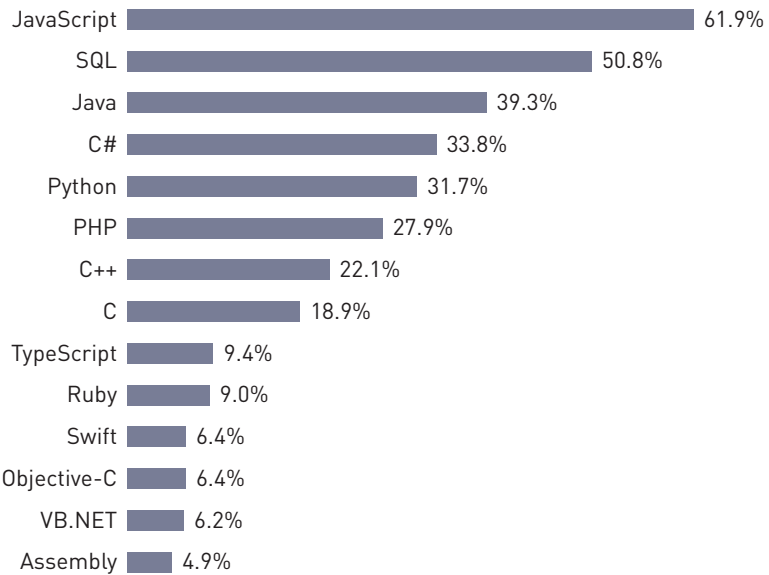


Figure 1.3.4 The percentage of programmers who use the most common programming languages

ACTIVITY 1.3.1

Different programming languages have strengths in certain areas. From figure 1.3.4, select the top seven languages and research the pros and cons of each, particularly what makes each one popular.

According to the Stack Overflow Developer Survey results, out of more than 64 000 developers globally who responded: ‘For the fifth year in a row, JavaScript was the most commonly used programming language. And once again, SQL takes second place, and Java third. However, the use of Python overtook PHP for the first time in five years.’



1.3.7
Developer
survey

ALGORITHM CONTROL STRUCTURES

It is possible to write any computer program by using just three basic control structures:

- 1 Sequence (step by step)
- 2 Selection (IF-THEN-ELSE)
- 3 Iteration, also known as repetition (DOWHILE).

Algorithms

An algorithm is independent from a computer programming language and consists of the logic and specific steps you develop to solve a computational problem. It is the unambiguity, structure and style of the algorithm that will determine the success of your final code. To take a practical example, suppose I gave you the following instructions in a house you had never been in before:

- a Turn left,
then turn right,
go down the stairs,
on your right enter the kitchen,
pick a cup, pour some hot water into it and add some hot chocolate.

- b Please make me a hot chocolate.

Example (a) is not ambiguous; it has clear, defined steps. In example (b), many steps remain undefined and I would likely not get my hot chocolate.

Algorithms allow us to represent our ideas in step-by-step procedures, in sequence – which, most of the time, is top down, like a recipe for baking a cake, and is independent of any programming language.

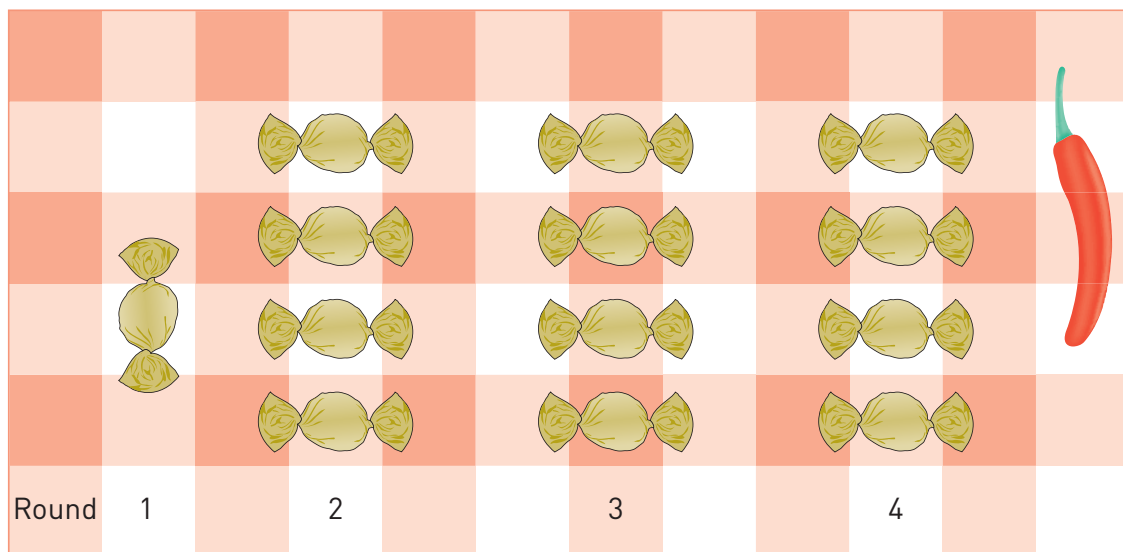


Figure 1.3.5 An algorithm holds the secret of winning the chilli game.

ACTIVITY 1.3.2

Requires: 13 chocolates, one chilli and two players

Rules: Players take turns to take one, two or three chocolates until there are no chocolates left, thus making the last player suffer the hot chilli.

Solution: There is an algorithm for this game that can guide you to choose the right number of chocolates based on what your opponent chooses, and that will ensure you never lose. The trick is to think in groups of four, as shown in figure 1.3.5. There are 13 chocolates in total. Taking one chocolate in the first round and four – (the number of chocolates the other player took in their most recent turn) in each subsequent round will ensure your opponent always eats the chilli. In groups, try to come up with a way to present a solution as an algorithm for this activity. Eventually you will be able to develop your very own 'Hot Chilli App' game.

The algorithm, combined with structured programming techniques, will guide the generation of the coded solution using a programming language. When the program is executed, it will control the electric flow in a smart device to perform a task. Humans solve problems in many ways, but in the world of digital

programming we need to problem-solve in a way that reflects how computer systems process data – that is, in a step-by-step, logical sequence. Smart devices are very good at:

- following step-by-step instructions – known as *sequence*
- repeating instructions thousands and millions of times – known as *iteration*
- dealing with decisions based on a true or false outcome – called *selection*. For example, if the light is red, then the car will stop.

Developing an algorithm

To develop and test an algorithm, some helpful guidelines are:

- 1 Explore and unpack the problem.
- 2 Develop an algorithm idea.
- 3 Desk check the algorithm.

Exploring and unpacking the problem

The first step in developing an algorithm is to clearly define the problem. Section 1.1 explored digital problems in a way that resulted in grouping inputs, process and output using an IPO chart. This is a good place to start. Let's take a simple example of adding four numbers and displaying the total. The digital problem might be broken down like this:

Step 1 Identify INPUTS and OUTPUTS

Underline all nouns and adjectives. This will normally identify the inputs and outputs. The problem statement would now look something like this:

A program is required to read four numbers, add them together and display their total.

Step 2 Identify PROCESS

Underline the verbs and adverbs, perhaps using a different colour, to identify the process. The problem statement would now look like:

A program is required to read four numbers, add them together and display their total.

Once the problem has been explored and unpacked in this way, develop an IPO chart.

Table 1.3.1 IPO chart adding four numbers

Input	Process	Output
number1 number2 number3 number4	READ four numbers ADD numbers together DISPLAY total	Total

We now clearly understand the input to the problem, the process of the problem and the output to be displayed.

Meaningful names

In this process, it is important to choose meaningful names for inputs and outputs as these names will be used later to represent variables or objects. A computer program will run happily regardless of the naming convention you choose for variables. Sometimes it is easier and faster simply to use A, B, C etc. as variable names. When a human being needs to understand your code, however, meaningful variable names are important. As already mentioned, a name given to a variable is simply a way to identify a storage location in memory. This is why it should be meaningful to all programmers who will read your code at a later stage. The uniqueness of the name will differentiate it from other memory locations. The name you choose should describe the type of data that will be stored in a particular variable. Variables can store numerical data such as an integer or a real number, or a character. In the previous example, number1, number2, number3 and number4



are more meaningful names than A, B, C, and D, and clearly represent the type of data that will be stored in memory. Using a naming convention is also useful for consistency. Different programmers use different naming conventions, which is unlikely to create problems provided each convention is logical and consistent.

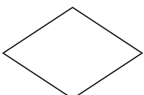
Developing an algorithm idea

Let's revisit the previous problem: 'A program is required to **read** four numbers, **add** them together and **display** their **total**.' We unpacked the problem into inputs, process and output, as shown in the IPO chart (Table 1.3.1). Using **pseudocode** and basic sequence control structure, the following solution is established:

```
Add_four_numbers
  DECLARE total
  DECLARE number1
  DECLARE number2
  DECLARE number3
  DECLARE number4
  number1 = INPUT number from user
  number2 = INPUT number from user
  number3 = INPUT number from user
  number4 = INPUT number from user
  total = number1 + number2 + number3 + number4
  OUTPUT total
END
```

'**Add_four_numbers**' is a name given to the algorithm. Names should briefly describe the function of the algorithm. 'END' indicates the algorithm has finished. Indentation has been used for readability between the beginning and the end. Most variables need to be declared. This means computer memory is made available to store data into this memory location for later use. 'number1', 'number2', 'number3', 'number4' and 'total' are all variables that are assigned data using the '=' symbol. Now that the algorithm is complete, it should be checked for errors by the use of desk checking. This will be explored later in this section.

Developing an algorithm requires knowledge of how to communicate algorithmic ideas. In this book, pseudocode such as the solution above is the preferred mechanism, but flowcharts can also be used. Pseudocode is not programming. It allows you to focus on the problem without worrying about syntax, and can be written using English statements. A flowchart is illustrated in figure 1.3.6.

Name	Symbol	Use in flowchart
Oval		Denotes the beginning or end of the program
Parallelogram		Denotes an input operation
Rectangle		Denotes a process to be carried out, e.g. addition, subtraction, division
		Denotes a decision (or branch) to be made. The program should continue along one of two routes (e.g. IF/THEN/ELSE)
Hybrid		Denotes an output operation
Flow line		Denotes the direction of logic flow in the program

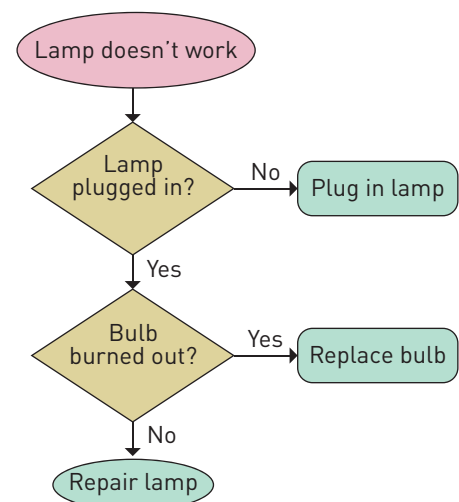


Figure 1.3.6 Flowchart symbols

Before developing pseudocode, an understanding of six basic computer operations that can be performed is required:

1 Arithmetic

A computer can perform mathematical calculations: + Add, – Subtract, * Multiply, / Divide, () parentheses.

2 Assignment

A computer can assign a value to a variable or memory location. Having variables gives you the ability to write one algorithm that can be used for many problems. In the first column of Table 1.3.2, there will only ever be one answer, but column 2 is flexible:

3 Sequence

A sequence is a number of instructions processed one after the other, as in figure 1.3.7.

4 Selection

The next instruction to be executed depends on a *condition*, as in figure 1.3.8. Indent instructions to be included under the selection statement to indicate scope of the statement. **IF**, **ELSE**, **ELSEIF**, **ENDIF** must be in line. For example:

DECLARE name, age, ageGroup

READ name, age

IF age > 19

 ageGroup = adult

ELSEIF age > 12

 ageGroup = teenager

ELSEIF age > 3

 ageGroup = child

ELSE

 ageGroup = infant

ENDIF

5 Condition

A logical expression that evaluates true or false, for example: 'age > 19'. We could use this condition to determine that the user is an adult if the condition is true.

6 Iteration (or loop)

A number of instructions repeated.

WHILE, **ENDWHILE** (or pretested loop) checks the condition before the loop starts. See figure 1.3.9 (right).

REPEAT, **UNTIL** (or post tested loop) checks the condition at the end of the loop. See figure 1.3.9 (left).

Modularisation

The two flowcharts in figure 1.3.9 demonstrate a very powerful structured programming technique known as modularisation. This is a process of deconstructing a large problem into smaller, more or less independent units or modules to allow for

PSEUDOCODE CONVENTIONS

Rules for pseudocode

Remember, you are describing your logic plan to develop a program; you are not programming!

- Write only one statement per line.
Computers can only deal with one instruction at a time.
- Capitalise keywords: INPUT, OUTPUT, IF, ELSE, ELSEIF, ENDIF, WHILE, ENDWHILE, DECLARE, CALL.
- Indent to show hierarchy, scope and structures.
- Keep statements language-independent.

Rules for variable names in pseudocode

- Begin with lower case.
- Contain no spaces.
- Additional words begin with capitals (camel case).
- Unique names within code.
- Consistent and meaningful use of names.

Table 1.3.2 Flexible vs. non-flexible algorithm

NON-FLEXIBLE ALGORITHM	FLEXIBLE ALGORITHM
WRITE 4 + 3	DECLARE answer DECLARE number1 DECLARE number2 number1 = INPUT number from user number2 = INPUT number from user answer = number1 + number2 OUTPUT answer

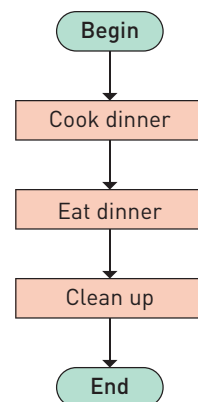


Figure 1.3.7 Sequence

the re-use of code and to reduce the complexity of a system, making it easier to find errors (desk check or debug). For example, should you want to animate the universe one day, it would be much easier and faster to create an algorithm for one star and re-use that to create a galaxy, rather than create an algorithm for every star. A module must contain multiple steps (instructions), as demonstrated above. This is achieved by placing two instructions into a module called 'process':

```
define process:
  ADD 1 TO count
  WRITE count
end process
```

Using modules to hide code is known as abstraction. Figure 1.3.10, showing an interactive face, highlights how much easier it is to read a program if you break instructions up into modules. You will also notice that, in this language, a module is known as a **function**. Compare the pseudocode on the left in figure 1.3.10 with that on the right.

Each instruction in the left column of figure 1.3.10 is a call to a submodule or function. In the right column, all the instructions are listed together, rather than in sub-modules. Breaking your problem up into modules makes it much easier to understand logic, easier to diagnose errors and easier to re-use instructions – and will save you a lot of time, as you will see in later sections. You can write one module for an eye, and simply call it with different X and Y coordinates, rather than write separate instructions for both eyes.

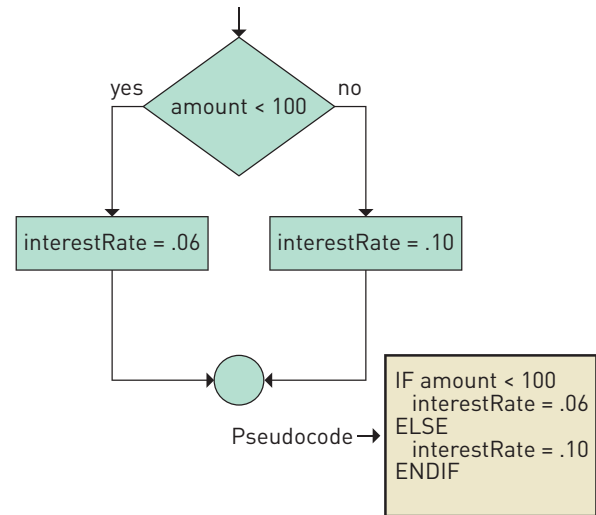
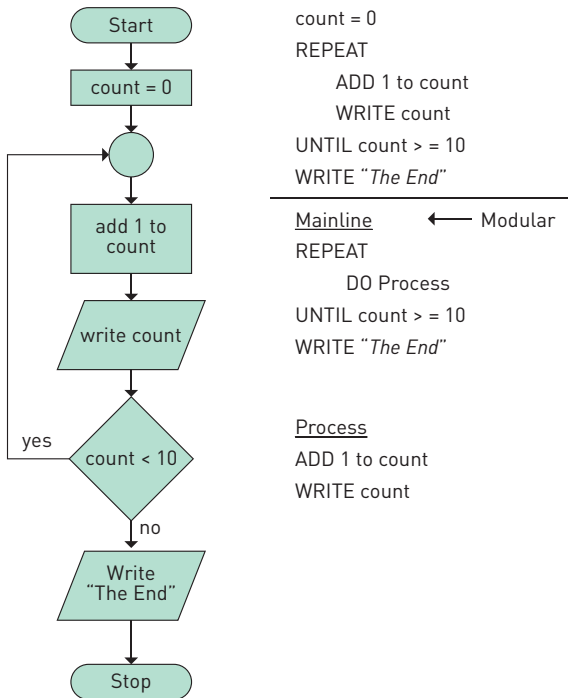


Figure 1.3.8 Selection in a flowchart

Repeat/Until



While/Endwhile

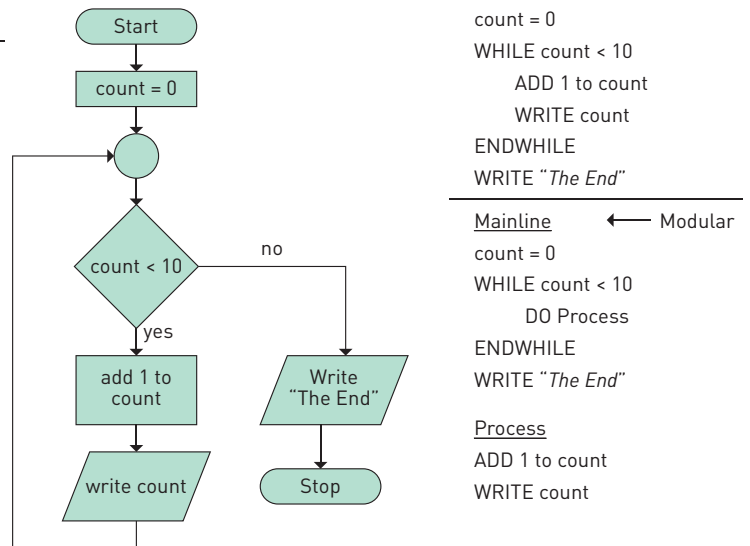


Figure 1.3.9 Post-tested iteration (left) and pre-tested iteration (right) in a flowchart

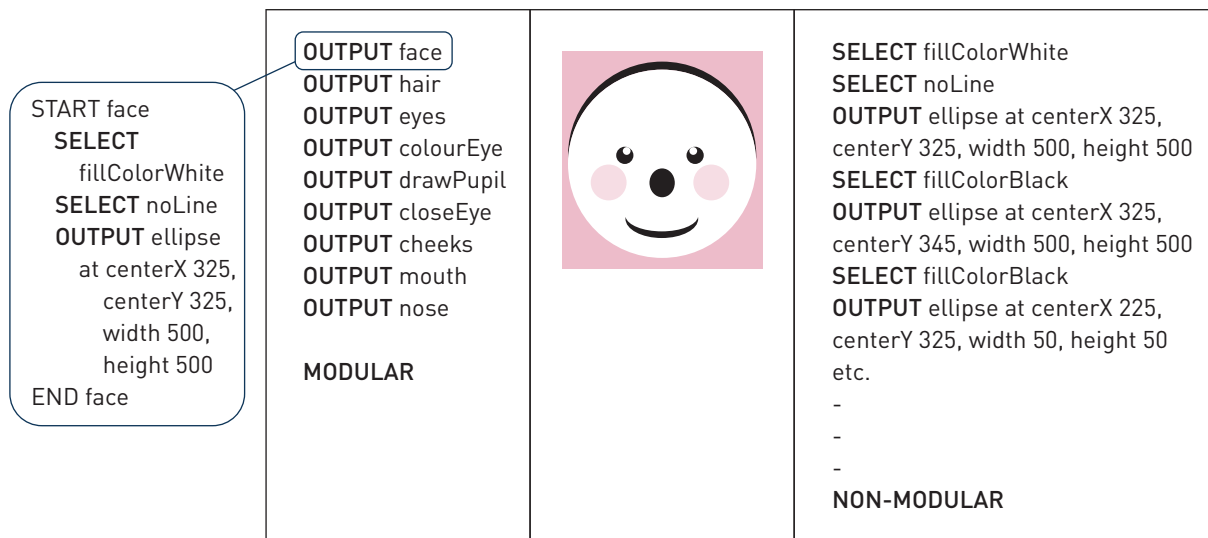


Figure 1.3.10 Modular and non-modular pseudocode

Desk check the algorithm

After the algorithm idea has been developed, it needs testing for correctness. This is called desk checking and is a necessary step as most major logic errors occur during the development of the algorithm. A logic error is present when the program runs fine but the output is incorrect. If the error is not detected at this stage, it will be passed on to the program. In the case studies below there are examples of desk checking. This involves tracing through the logic of the algorithm with some chosen test data. That is, you walk through the logic of the algorithm exactly as a computer would, keeping track of all major variable values on a sheet of paper. A trace table is often used; this is demonstrated in the case studies.

Selecting test data

To desk check the algorithm, you need only a few simple test cases that follow the major paths of the algorithm logic.

DESK CHECKING

These are the simple steps to follow when desk checking an algorithm:

- 1 Choose two or three simple input test cases that are valid.
- 2 Ensure you have the correct results for each test case.
- 3 Make a trace table with relevant variable names in each column.
- 4 Walk through the first test case, recording the step-by-step values of the variables.
- 5 Repeat this for each set of test data and ensure correct results are output.

Let's now check the Add_four_numbers algorithm.

STEP	NUMBER1	NUMBER2	NUMBER3	NUMBER4	TOTAL
1	10	20	30	40	100
2	1	2	3	4	10

The test data is in each column and, since all variables were assigned values and totalled in the one step, there is only one line in the trace table. Two tests were performed, resulting in two lines. Normally, when there is looping involved, more rows will appear in the table.

ACTIVITY 1.3.3

- 1 Write pseudocode for an algorithm to output the first five odd numbers using sequence only.
- 2 Write pseudocode for an algorithm to do the same as question 1 using repetition.
- 3 Write pseudocode for an algorithm that accepts the input 'odd', which prints out the first five odd numbers, or 'even', to output the first five even numbers.
- 4 Write pseudocode that will do the same as question 4 but uses a module to complete even numbers and a module to complete odd numbers.

Brief explanation of HTML

Before you dive into your first scenario, it is well worth refreshing your understanding of HTML (Hypertext Markup Language), CSS (Cascading Style Sheets) and DOM (Document Object Model) given that you will no doubt face this at some stage in your programming career. If you are using Python, Javascript or any other high-level language, you still need a basic understanding of HTML, CSS and DOM to make any web-based app. HTML is a set of tags used to display content on a web browser. HTML has no other purpose but to mark up content into a web browser – hence the name. All web browsers are capable of interpreting the meaning of HTML tags.

Our next few 'walk-throughs' rely on basic HTML knowledge. The Internet has many free resources that you can go to for a basic understanding of HTML, such as the W3Schools website. Web pages are held together using HTML. Your HTML file is a simple text file, with the file extension .html, that is interpreted by web browsers – for example, Mozilla Firefox, Microsoft Internet Explorer (more recently known as Microsoft Edge), Google Chrome and Apple Safari.

ACTIVITY 1.3.4

Create a folder in your device and call it 'firstweb'. Type the code below into Notepad or a simple text editor:

```
<!DOCTYPE html>
<html>

<head>
  <meta charset="UTF-8">
  <title> insert title here </title>
</head>

<body>

</body>

</html>
```

Save it as template.htm and open it in a web browser, which should display a blank page. In the text editor, add a title to the page and some text in between the body tags, and save it. Refresh your browser to see the change. Every change you make in your HTML file, once saved, will be seen in the web browser once you refresh.

There are lots of free editors, including Adobe Brackets, Atom and Sublime, that provide many coding tools. One free lightweight editor that will be used throughout this section is Visual Studio Code (VSC). VSC allows you to download libraries that provide you with predictive help known as 'Intellisense' when coding in many languages, including HTML. It also performs automatic formatting and debugging. For more information about these editors, you can go to their websites, where you will find many resources to get you up and running.



Algorithm walk-throughs

The following two examples, Fibonacci bubbles and Euclid's tiles, demonstrate the power of the algorithm. Each walk-through will demonstrate the use of algorithms and the explore, develop, generate, evaluate (EDGE) approach to learning and problem-solving, introduced in section 1.1.

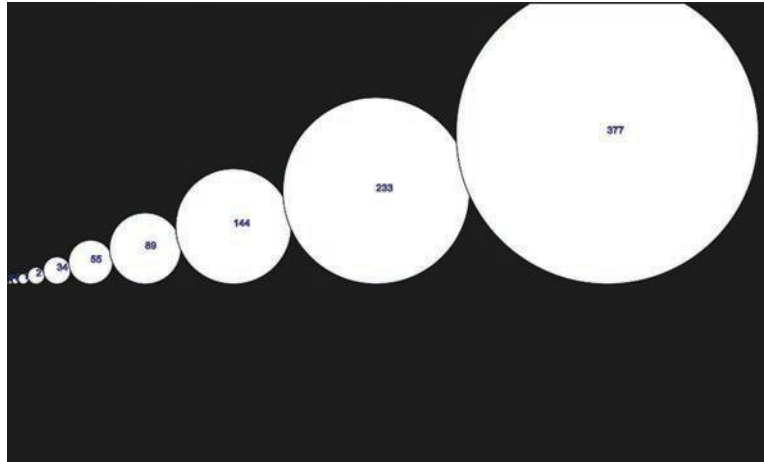


Figure 1.3.11 Empty example in Visual Studio Code (VSC)



Fibonacci bubbles

Let's explore the Fibonacci bubbles problem.

Restate the problem succinctly

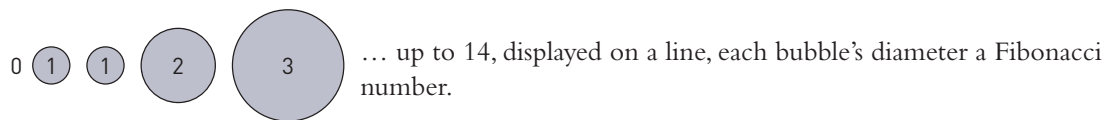
The task is to make 14 Fibonacci bubbles.

Analyse the human need or want

- 1 What is the Fibonacci sequence?

The Fibonacci sequence is characterised by the fact that every number after the first two is the sum of those two preceding numbers: 0, 1, 1, 2, 3, 5, 8, and so on.

- 2 How can bubbles be used to visualise this?



Decompose the problem into smaller parts

The table below demonstrates an IPO chart as a good way to explore the starting point for algorithms.

Input (data)	Process (algorithm)	Output (information/animation)
Store first two Fibonacci numbers using variables	Algorithm to: loop to calculate up to the 14th Fibonacci number calculate diameter of each bubble and centre points	14 bubbles horizontally (X direction) with each bubble diameter matching that Fibonacci number and the number displayed in centre of bubble

Identify requirements

In order to avoid having to physically draw and calculate the area of 14 Fibonacci bubbles, an algorithm is needed that will calculate the sum of the two previous numbers up to an n th number.

There are many different programming languages and integrated development environments (IDE) you may decide to use. Refer back to section 1.1 for more information on this.



Recognise constraints

The width of the browser will limit how many Fibonacci bubbles can be displayed.

In this case we will limit the number of bubbles to 14.



Develop the algorithm idea

DECLARE variable numberOfBubbles as integer	//n is the number of bubbles
DECLARE variable sum as integer	//sum of the two previous numbers
DECLARE variable diameter as integer	//diameter of the fib bubble
DECLARE variable f1 as integer	//first number of the previous pair
DECLARE variable f2 as integer	//second number of the previous pair
sum = 0	
f1 = 0	//set first number to 0
f2 = 1	//set the second number to 1
numberOfBubbles = 0	//set the number of bubbles to 0
WHILE numberOfBubbles < 14	//loop until 14 bubbles
sum = f1 + f2	//add previous two numbers
f2 = f1	//assign the second number the value of the
	//first num
f1 = sum	//assign f1 the value of sum
diameter = sum	//assign the diameter of the bubble the fib
	//number
	//increments to next bubble
ADD 1 to numberOfBubbles	//draw bubble
WRITE bubble with diameter and Xcenter,Ycenter	
ENDWHILE	

In the example above, indentation was used to indicate scope of the iteration. A flowchart could also be used to represent your algorithm.

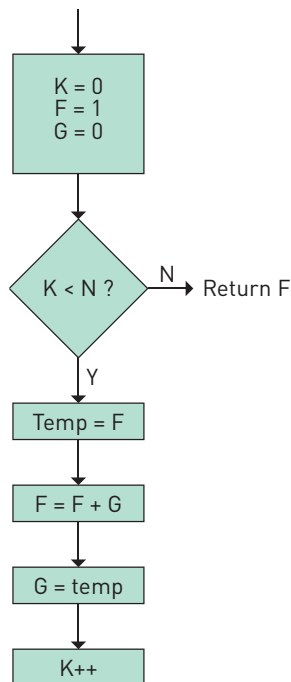


Figure 1.3.12 Flowchart example

Desk check

Table 1.3.3 represents the check table or trace table containing the data as we step through the sequence of instructions in the algorithm. We loop through the algorithm in the same way that the computer would, and record the results in each row below, in order from the top line to the bottom.

The trace table stops at bubble 6 as it is clear by then that the Fibonacci sequence is being calculated. Desk checking is examined in more depth in section 1.4.

Develop ideas for the user interface

In terms of user interface design, a 400×400 canvas will be used on a web page, and bubbles go horizontally along the bottom with diameters of the Fibonacci number.

Generate the program

Before a solution can be generated, it is important to understand the syntax (set of rules) of the programming language chosen. In this case the chosen framework is p5.js, which has a particular program flow and easy-to-learn coding. It uses two global functions called 'setup()' and 'draw()'. The setup function runs once at the start of the program and the draw function continues looping for the duration of the program. This lends itself nicely to animation and interactive visual displays in the web browser.

Table 1.3.3 Trace table

NUMBER OF BUBBLES	SUM	DIAMETER	F1	F2	
0	0	--	0	1	Input from assigned values
1	1	1	1	0	Loop1
2	1	1	1	1	Loop2
3	2	2	2	1	Loop3
4	3	3	3	2	Loop4
5	5	5	5	3	Loop5
6	8	8	8	5	Loop6

Step 1: Set up the user interface

This Fibonacci bubbles program runs in a web page so you will need to set up a basic HTML page and a JavaScript file that links to the p5.js library. Figure 1.3.13 shows an index.html, test.js and a p5.js file in the file explorer, and the VSC p5.js canvas extension that allows you to preview your sketch. An ellipse has been entered to test the layout is working. The test.js is shown in figure 1.3.14 and will be used as a test file for this Fibonacci bubbles example. You can rename it more meaningfully later, but take care if you do so to also update the links in your index.html file. The basic index.html file is like the ‘glue’ that links all three files together. To link an external JavaScript file to the HTML file, you use the <script> tag (see the HTML code below).

```
<!DOCTYPE HTML>
<html>
  <head>
    <meta http-equiv="Content-Type" content = "text/html; charset = UTF-8"/>
    <title>Your Website</title>
    <script src="p5.js"></script>
    <script src="test.js"></script>
  </head>
  <body>
    <h1>Fibonacci bubbles</h1>
  </body>
</html>
```

In HTML it is only the content between the <body> tags that will appear in the web browser. The tags in the <head> section are either linking to other libraries such as Google Fonts or meta-information about the site such as search words and page title. HTML tags and CSS must obey a set of rules known as DOM (Document Object Model).



1.3.9
Introduction
to DOM

Step 2: Enter the algorithm for Fibonacci

Since the draw() function loops forever, we have no need for a loop control structure. We do, however, need to stop making bubbles at 14 with the use of selection (IF), shown in figure 1.3.14. The X and Y position of the bubbles and text needed for the Fibonacci number display are relative to the diameter.

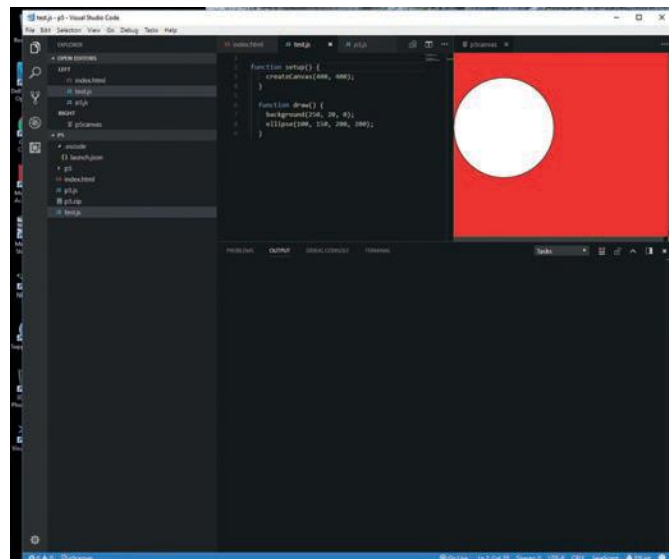


Figure 1.3.13 Starting Fibonacci bubbles example in VSC

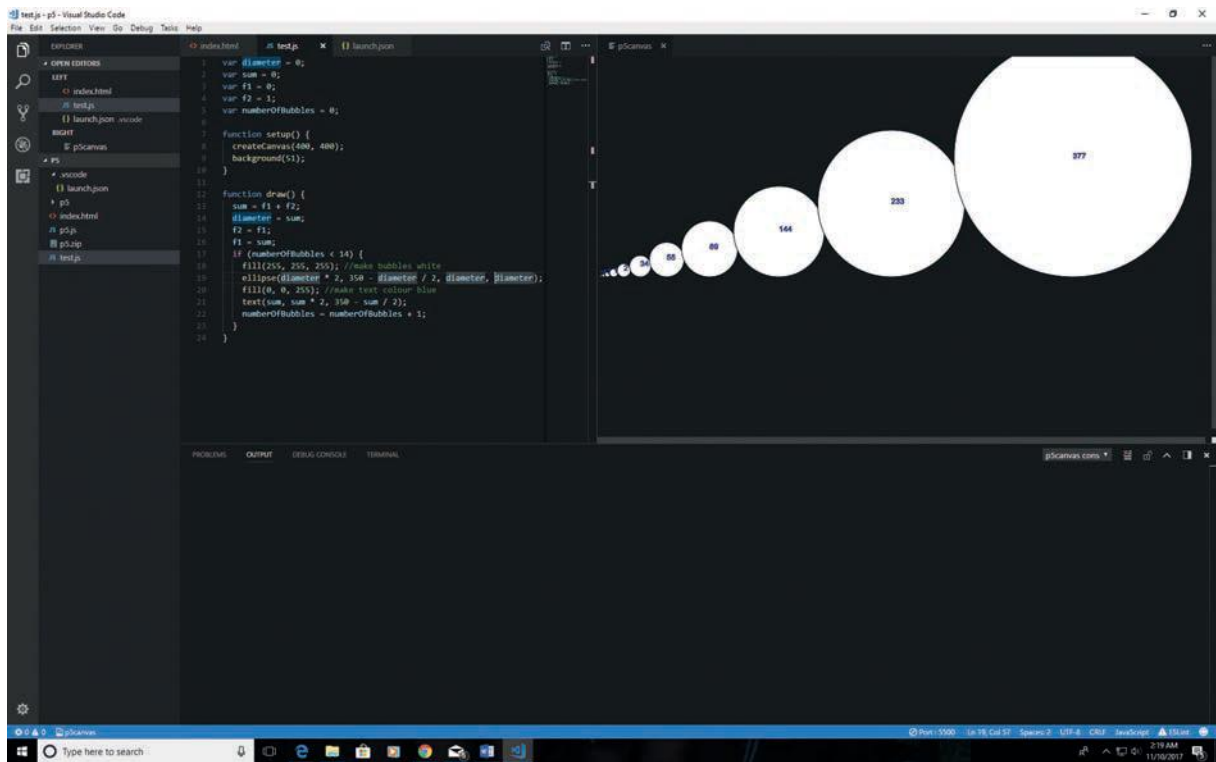


Figure 1.3.14 The user interface constrains the number of bubbles that can be displayed.

Euclid's tiles

A tile company has approached you to write an algorithm that will help them to order the largest square tile for a floor measuring $1.5 \text{ m} \times 3.45 \text{ m}$ that ensures there will be no need to cut tiles and so avoids wastage. Your final program will visualise the result in a browser.

Summarise the task:

- Write an algorithm to find the largest square tile to fit perfectly in an area $1.5 \text{ m} \times 3.45 \text{ m}$.
- Visualise the layout in the browser.
- The space needs to be divided up so that there is no wastage.

Table 1.3.4 shows the initial exploration of inputs, process and outputs.

Table 1.3.4 IPO chart

INPUT (data)	PROCESSING (algorithm)	OUTPUT (information/animation)
Dimensions of the floor – in this case, $1.5 \text{ m} \times 3.45 \text{ m}$	Algorithm to: find the largest square tile to fit perfectly in an area $1.5 \text{ m} \times 3.45 \text{ m}$	Output the dimensions of the largest square tile that will fit perfectly into the floor area Output visual display of this



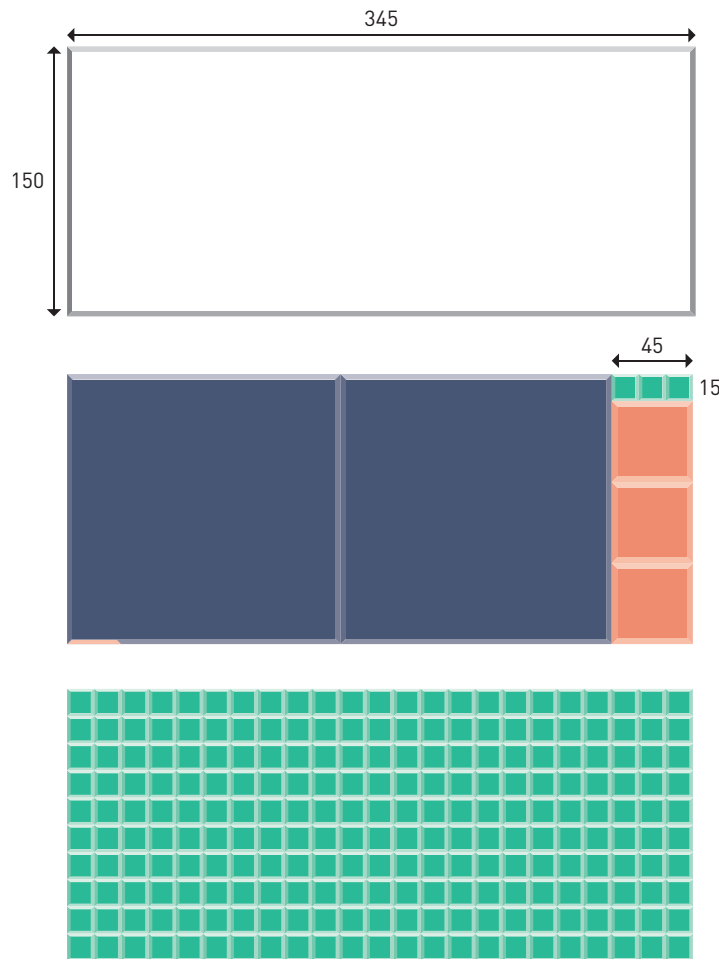


Figure 1.3.15 Sketching the floor

Euclid developed an algorithm to calculate the ‘greatest common divisor’ of two numbers. The algorithm is freely available on the Internet, but coding it yourself is more of a challenge.

If we take the width to be 150 cm and the length to be 345 cm, then Euclid’s algorithm finds the greatest common divisor of both dimensions. This then would be the dimension of our perfectly sized square tile. How did he do this? Geometrically, this could be described as follows:

- Step 1** Fill as much as possible of the area with square tiles of dimensions that are the same as the smallest side of the larger rectangle (see blue tiles in figure 1.3.15).
- Step 2** Repeat step 1 in the remaining space (see orange tiles in figure 1.3.15).
- Step 3** Repeat step 2. The green tiles show there is no more room for any more tiles and there is no remainder. The green tile is the perfect size (15 cm square), which is calculated by dividing 45 by 3 (see figure 1.3.15).

In order to develop an algorithm that will account for any floor dimensions, you will need to create variables to store values. It is assumed the width is always the shortest side. Many of the variables created in the algorithm below are needed to display the tiles on the screen.

Once the algorithm is detailed enough, you can copy and paste the algorithm into your IDE as comments and slowly change each algorithm step to the correct programming syntax in the programming language you plan to use. Commenting is the process of using a symbol to tell the compiler/interpreter to ignore human notes and instructions. In the code below, the symbol ‘//’ is used for comments, but this will vary from language to language. Commenting out is also a useful technique when debugging, as it can be used to isolate code that you know works.



Develop the algorithm based on our exploration above. Steps 1–3 indicate that you need to loop the process of dividing the shortest side into the longest side until there is no remainder. This can be done through coding.

Table 1.3.5 Algorithm to solve the Euclid's Tiles problem

Top-level variables are called global variables and are available to all modules

Local variables are variables that are only used in a particular module

The '//' symbol indicates a program comment to help readers understand the purpose of the instructions

A rectangle function has 'rect(X,Y,W,H)' and 'X,Y' are the coordinates of the starting point, which is taken to be the top left corner of the rectangle

The + symbol when outputting messages is used to join variables and the message called concatenation

TOP LEVEL	SECOND LEVEL
DECLARE floorWidth	calculateTile
DECLARE floorLength	INPUT floorWidth
DECLARE tileWidth	INPUT floorLength
DECLARE tileLength	tileWidth = floorWidth
DECLARE remainder	tileHeight = floorHeight
DECLARE numberOfTilesX	remainder = length MOD width //MOD is used to represent the remainder of a division
DECLARE numberOfTilesY	WHILE remainder > 0
CALL calculateTile	length = width
CALL displayFloor	width = remainder
CALL displayTileSize	remainder = length MOD width
	ENDWHILE
	tileWidth = tileLength
	displayFloor
	DECLARE countTilesX //this is a local variable to store the number of tiles in a row
	DECLARE countTilesY //to store the number of tiles in a column
	OUTPUT background width 400 height 400
	WHILE tilesY < numberOfTilesY //stops drawing past the column
	IF tilesX < numberOfTilesX //stops drawing past the box
	OUTPUT tile(0 + tilesX) * tile_width, (0 + tilesY) * tile_length, tile_width, tile_length
	//draw the tile
	countTilesX = countTilesX + 1 //increments the tile count when a row tile is drawn on the screen
	ELSE
	countTilesY = countTilesY + 1 //increments the tile count when a row is full
	countTilesX = 0 //restarts the next row count
	ENDIF
	ENDWHILE
	displayTileSize
	OUTPUT "Your perfect square tile has a dimension of " + tile_length + " cm square.", X pos 50, Y pos 200]
	OUTPUT ("You need " + numberOfTilesX * numberOfTilesY + " tiles. ", X pos 50, Y pos 220);

The user interface in this simple scenario will be generated algorithmically. When developing visual ideas on how to solve this problem, you need to think in terms of iterations. The first iteration draws row 1 boxes as shown in figure 1.3.15, then completes row 2 and continues until the rows in the Y direction are complete. This will give you an idea of how the algorithm could be designed, as in the module 'displayFloor' in Table 1.3.5 above. The term 'tilesY' keeps track of the rows of tiles completed and the line 'WHILE tilesY < numberOfTilesY' ensures the algorithm will loop until all rows of tiles are drawn.

ACTIVITY 1.3.5

Draw a trace table and complete a desk check on the first row of the module 'displayFloor'. A clue is to count up all the variables involved in the module and add them to the top row of your table. Try to keep them in order of how they appear in the pseudocode. Sketch each tile on paper to see if you get the same result.

Using the same IDE and framework as the Fibonacci bubbles, the following output and code evolved from our pseudocode above.

```

/* This program uses Euclid's
algorithm to find the perfect size
square tile for a given room */
//global variable declarations
var floor_width;
var floor_length;
var tile_width;
var tile_length;
var remainder;
var numberOfTilesX;
var numberOfTilesY;

//initialise global variables
numberOfTilesX = 0;
numberOfTilesY = 0;
floor_width = 150;
floor_length = 345;
tile_width = floor_width;
tile_length = floor_length;

//this function runs once only
at start of program
function setup() {
  createCanvas(400, 400); //creates a canvas
  background(122, 151, 100); //creates a background
}

//this function normally runs continuously throughout the duration of the program
//in this example I have turned off looping automatically to demonstrate
other looping control structures
function draw() {
  noLoop();
  calculateTileSize();
  displayFloor();
  displayTileSize();
}

/* This function uses Euclid's algorithm to find the perfect size square
tile for a given room */
function calculateTileSize() {
  remainder = floor_length % floor_width;
  while (remainder >= 0) {
    tile_length = tile_width;
    tile_width = remainder;
    remainder = tile_length % tile_width;
  }
  tile_width = tile_length;
}

/* This function displays the tiles onto the browser for a room of a given
size */
function displayFloor() {
  var countTilesX;
  var countTilesY;

```

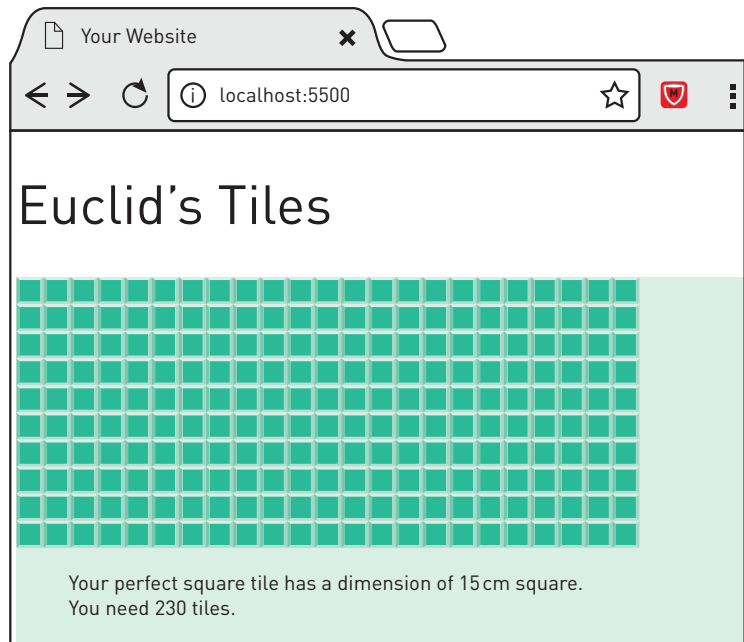


Figure 1.3.16 Final output of Euclid's tiles

```

//initialise variables
countTilesX = 0;
countTilesY = 0;
numberOfTilesX = floor_length / tile_length; //stores number of tiles in
//a row
numberOfTilesY = floor_width / tile_width; //stores the number of
//tiles in a column

strokeWeight(2);
rect(0, 0, floor_width, floor_length); //output the background

//generate tiles
while (countTilesY < numberOfTilesY) { //prevents drawing over in a column
  if (countTilesX < numberOfTilesX) { //prevents drawing over in a row
    rect((0 + countTilesX) * tile_width, (0 + countTilesY) * tile_length,
    tile_width, tile_length); //output the tile
    countTilesX = countTilesX + 1;
  } else {
    countTilesY = countTilesY + 1;
    countTilesX = 0;
  }
}
}

/* This function displays the number of tiles and tile size dimension onto
the browser */
function displayTileSize() {
  text("Your perfect square tile has a dimension of" + tile_length +
  "cm square.", 50, 200);
  text("You need" + numberOfTilesX * numberOfTilesY + "tiles.", 50, 220);
}

```

ACTIVITY 1.3.6

Add two simple input boxes to the Euclid's tiles program that take any width and height of a room from the user and then generate the tiles.

As we progress through each scenario, it will become clear that effective programming or coding requires an understanding of:

- variables, control structures and data structures
- syntax of the language and keeping up to date with changes
- an Integrated Development Environment (IDE) for authoring and debugging.

A brief history of programming languages

From the early days of programming, higher-level languages such as Fortran (1957) and Cobol (1959) were developed to let people write code that was independent of any specific computer. As ideas grew and new ways to express code developed, along came new programming languages, such as Basic, Pascal, C, C++, Python and other high-level languages including Java. Python was created as a server side language and has grown in popularity due to its data-handling capabilities – for example, the NumPy and Pandas modules.

Web app development relies on the web-based programming language JavaScript. In 1995, James Gosling at Sun Microsystems developed Java as a core component of the Java platform to deliver web applets to the browser. JavaScript was created over 10 days in May 1995 by Brendan Eich, then working at Netscape and now of Mozilla, in the hope of capitalising on the success of Java. JavaScript was not always known as JavaScript: its original name was Mocha, a name chosen by Marc Andreessen, founder of Netscape. Netscape then submitted JavaScript to the European Computer Manufacturers Association (ECMA) for standardisation. ECMA is an organisation that develops and promotes standards in information and communication technology. This resulted in a new language standard, known as ECMAScript, or ES for short. ECMAScript was designed by a technical committee of ECMA (TC39), which meets regularly. Minutes of the meetings are available online and give you



Python works across many platforms and has a large number of code modules others have written for use.

a good idea of how TC39 works. It is interesting to note that TC39 operates by consensus: decisions require that a large majority agrees and nobody disagrees strongly enough to veto. For many members, agreements lead to real obligations (they'll have to implement features etc.) JavaScript happens to be the most popular implementation of ES. Every time you see ES followed by a number (for example, ES2017 or ES8), it is referencing an edition of ECMAScript.

In fact, there have been eight editions of ECMAScript published between 1997 and 2017. Standards are changing rapidly and adding the year makes the version name more meaningful. Another high-level language, Processing, was built on top of Java in 2001 by Casey Reas and Ben Fry. Both of them are designers, visual artists and programmers, and they needed a language for expressing creative visual animations and creative application development.

Since then, JavaScript has become more popular because it works directly in a browser and is shared instantly with friends. In 2015, an artist named Lauren McCarthy identified this trend and began a project in cooperation with the Processing foundation to have an easy-to-learn JavaScript framework that would do the job for which Processing was originally designed, but for the web. This framework took the name p5.js. It differs from Processing in that it uses JavaScript, not Java, behind its framework. Along with Python, JavaScript has many frameworks, and libraries contained within these frameworks, that work across many platforms. These include BBC Microbits, Lego Robotics, automating drone flight, front-end and back-end web development with Node.js, data visualisation, mobile development, websites and much more. With any web app, you also need to understand HTML and CSS. Programming language versions, frameworks and contained libraries and modules are continually changing and improving, but the efficient and effective algorithm still remains the key to successful code.

Basic structures of programming

We have already discussed the basic control structures for looping (iteration) and decisions (selection) and the data structure known as a variable. We will recap what we have already discussed and then extend our knowledge to more complex data structures.

ACTIVITY 1.3.7

Create a PMI map or infographic of historic and available programming languages.

Control structures

Looping (iteration)

- 1 Counted loop – also known as a **FOR loop**. Used for looping when you know the exact number of times you need to loop.
- 2 Pretested loop – also known as a **WHILE loop**. Used for looping until a condition is met at the beginning of the loop.
- 3 Post tested loop – also known as a **REPEAT loop**. Used for looping until a condition is met at the end of the loop.

Selection

- 1 **IF statement** – used for selection based on a condition being true or false.
- 2 **Nested IF statements** occur when you need a condition within a condition. For example:


```
IF infrared sensor reads bright light THEN
  IF temperature sensor reads hot THEN
    UV rating is high
  END IF
END IF
```
- 3 **Logic operators** – can also be used in situations where two or more conditions need to be met. For example:


```
IF infrared sensor reads bright AND temperature sensor reads hot THEN
  UV rating is high
END IF
```


Data structures

Variables

Variables are used for storing data. Data stored in a programming language has a name and a data type. Variable names need to be meaningful and detailed enough to indicate the purpose of the data. Common data types are characters or text, numbers and Boolean values. The data type is important because different binary encoding schemes are used for different data types. For example, alphanumeric data is coded using ASCII codes that match a key on the keyboard, while numerical data is coded as a base 2 version of a decimal number. Operations between variables behave differently depending on the data type. For example, $2 + 2 = 4$ when working with numerical data, but $'2' + '2' = '22'$ when working with alphanumeric data. To perform mathematical operations, alphanumeric data needs first to be converted to numbers.

Arrays

An array is used when more than one value of the same type needs to be stored. For example, imagine we needed to sort the top five marks in the class. The array could look like figure 1.3.17.

mark[0] mark[1] mark[2] mark[3] mark[4]

19	10	8	17	9

Figure 1.3.17 Array data structure

The array name in figure 1.3.17 is 'mark' and each element of the array can be accessed by a pointer, also known as an index. The first data element, '19', can be accessed by using the array name and the pointer value in square brackets: 'mark[0]'. Array pointers nearly always start at 0. An array has a length, which in this case is 5, and the data type stored in this array is a number. The original question asks us to sort these from lowest to highest mark.

ACTIVITY 1.3.8

Explore sorting algorithms and the library of functions within the framework you are using. We want to know more about arrays. In this example, using the JavaScript framework, type 'mozilla foundation array javascript' into your search engine.



Bubble sort

Bubble sort is the simplest sorting algorithm. It works by repeatedly swapping adjacent elements if they are in the wrong order. In the same way that bubbles float to the surface, the largest bubble gets there first.

When developing the algorithm, you would end up with pseudocode something like this:

```

DECLARE marks = 5 //marks stores the number of marks to be sorted
REPEAT
  flag = false; //flag to show true when 1st mark > 2nd mark
  FOR pointer = 0 to marks - 1 DO //counted loop that continues until end of array
    IF mark[pointer] > mark[pointer + 1] THEN //test if current mark > next mark
      swap the records //swap can be made into a submodule

    flag = true

  END IF
END FOR
  marks = marks - 1; //keep track
UNTIL flag = false or marks = 1 //stops when numbers are sorted or last array element
  
```



1.3.10 Java
script sorting
arrays
1.3.11 Sorting

ACTIVITY 1.3.9

Consider why the counted loop stops at pointer = marks - 1.

Consider why the repeat loop continues until flag = false or marks = 1.

In the develop stage, you should test your algorithm using a desk check, as demonstrated earlier in this section. Complete a desk check to ensure the algorithm will have no errors. Then generate the code from this algorithm.



The errors that you identify when testing your algorithm in the develop stage are known as logic errors. This means that the code is likely to work, but it will give you an incorrect outcome. When you test your code in the generate phase, you are more likely to identify syntax errors. These errors will stop your program from running as you have made a mistake in the syntax of the programming language you are using.

Objects

Objects are used in the ASCII Aliens game, introduced in section 1.1. **Objects** are a way of organising data (properties) and functionality (methods) into an advanced data structure. The following is an example of pseudocode for processing student information in a school administration system, with and without objects:

```
DECLARE studentAge = 14
DECLARE studentName = Mary
DECLARE studentSubjects = [Digital Technology, Maths, English] //an array of subjects
DECLARE studentBirthCity = Cairns
DECLARE BirthState = Qld

//output student details
OUTPUT "All about" + studentName
OUTPUT studentName + " is " + studentAge + " years old"
OUTPUT studentName + " studies " + studentSubjects[0] + "," + studentSubjects[1] + "," +
studentSubjects[2]
OUTPUT studentName + " was born in " + studentBirthCity + "," + studentBirthState
```

We can use the '+' symbol to add numbers and we can also use it to join strings together (concatenate), as seen in the example above.

If we had to process many students, the logic above would not be efficient. It would be much easier to use one structure to store the student data. This is where the object data structure can be used. Objects reflect a real world structure – for example, cars have properties such as colour, make and model. The object data structure is based on a class. A class is like the blueprint or template that we first define that encapsulates all the properties and methods that describe what it is we want to create, such as a certain species of animal. The process of creating a class of properties and methods is called **encapsulation**. A class **constructor** is used to define the properties or data in a class. Methods or functions are also created for the functionality of a class. The advantage of using a class is that it can be re-used and inherited into other classes and subclasses known as child classes. In the pseudocode below you can see the class constructor defines the properties. In this example there are no methods. Once the class data has been constructed and any methods created, an object can be instantiated (created and called upon), as in the following example:

```
DECLARE student class { //class constructor
    age: 14 //class properties
    name: Mary
    subjects: [Digital Technology, Maths, English] //property stored in an array
    birthplace: { //example of nested properties
        city: Cairns,
        state: Qld
    }
}
```

```
//output student details
```

```
OUTPUT "All about" + student.name
```

```
OUTPUT student.name + " is " + student.age + " years old"
```

```
OUTPUT student.name + " studies " + student.subjects[0] + "," + student.subjects[1] + "," + student.subjects[2]
```

```
OUTPUT student.name + " was born in " + student.birthplace.city + "," + student.birthplace.state
```

ACTIVITY 1.3.10

- 1 From the example above, identify the difference between how variable data is accessed compared to object data stored in object properties. Discuss in a group and compare your findings.
- 2 Birthplace data is stored within a nested property within an object. How is that data accessed?
- 3 In your own development environment, generate the code for the above example, run and test it.
- 4 Try creating a car object with properties and then display those properties on the screen.

Object properties and methods are accessed with the use of a period (full stop) symbol – for example, ‘objectname.property’ and ‘objectname.method’. Note that the example above uses properties but does not contain any methods.

Event listeners

Another way to control the flow of your program is through the use of event listeners. These are functions that are called when a particular event occurs – for example, when a mouse is pressed. In figure 1.3.18, when the mouse is pressed on the canvas (right), the word changes to the next element in the words array.

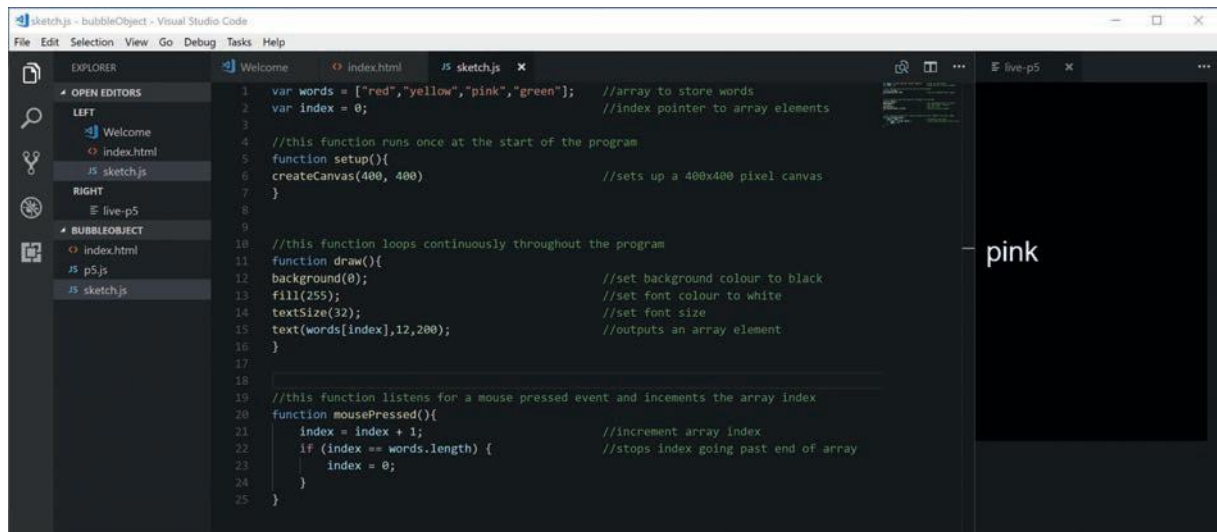


Figure 1.3.18 Mouse-pressed event listener

ACTIVITY 1.3.11

In the function ‘mousePressed’ in figure 1.3.18 there is an if statement. Investigate the purpose of this selection structure and discuss as a group. Generate this program in your own development environment. Explore the debugging tools and trace the code, watching the index variable change on each mouse press.

Debugging tools are a vital part of your arsenal to find programming errors. When you choose your programming environment, you should also become familiar with the debugging tools. In figure 1.3.18 the programming environment Visual Studio Code (VSC) is used. It is equipped with debugging tools that allow you to watch the value of variables change as you step through the program. In the debugging tools you can open a watch window that displays the values. To step through the code you need to add a breakpoint that will pause the program until you step forward one line at a time. Figure 1.3.19 demonstrates how the Chrome debugging extension for VSC allows you to preview the applications while using the debugging tools. On the left, the index variable has been added to the watch window and you can see the value is 0. The red dot next to line 21 indicates a breakpoint where the program paused once the browser was clicked on.

Formatting such as indentation in your code is important for readability – and in some languages, such as Python, it is compulsory – as it determines scope of control structures. In VSC, you can download an extension called Beautify Code for JavaScript that automatically formats your code on save. Once you download and install it, you need to go into preferences and turn ‘auto format on save’ to ‘true’. Beautiful code that follows indentation conventions and has detailed comments and meaningful variable names is much easier to debug.

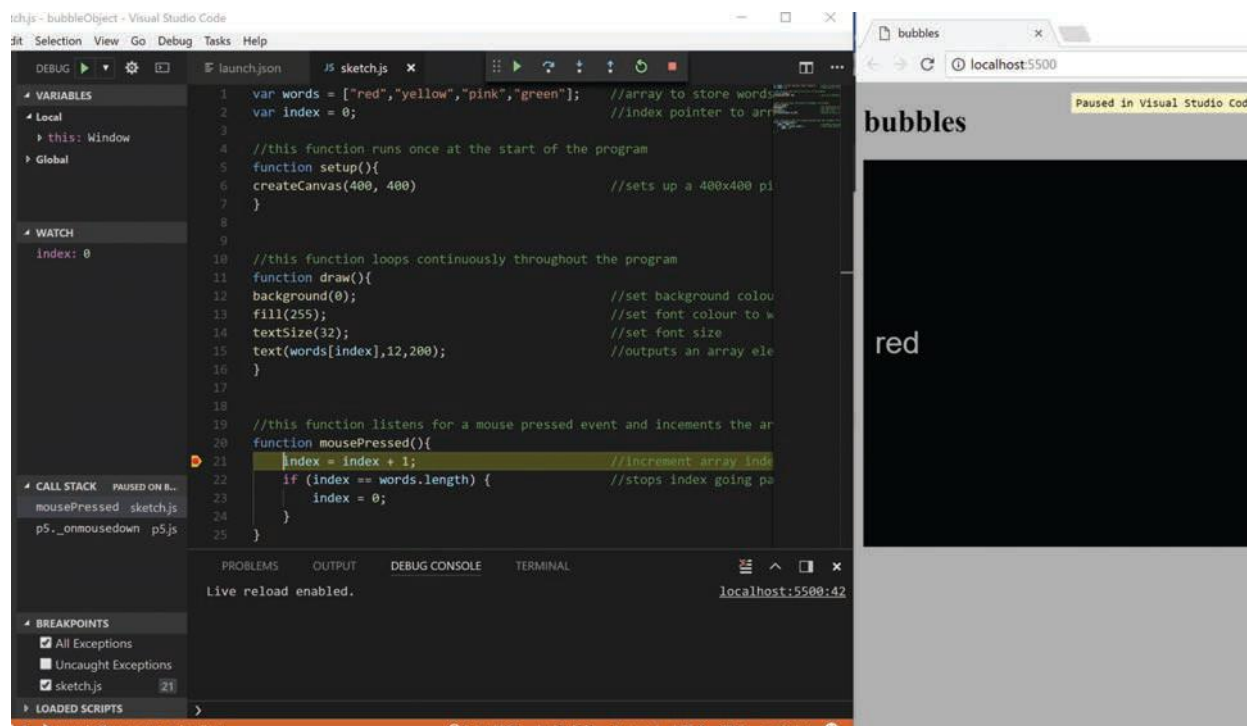


Figure 1.3.19 Debugging tools

Array of objects example

Let's walk through an example that uses a mouse event listener, an array and objects to further understand the array and object data structures: ‘When I click the mouse on the canvas in the browser window a bubble should appear, drop and bounce.’

Using the Digital Solutions problem-solving process, you would first explore the problem in more detail. Using an IPO chart, as we did earlier, we would have:

Input	Process	Output
Mouse click	Bubble X and Y position Bubble speed Bubble direction Bubble movement	Bubble on screen



There are many bubbles that need to be stored, and an array is a perfect data structure for this. As you explore the nature of this problem, you would most likely decide that each bubble has common properties such as size, colour, an X,Y position on the screen and a speed, and that the bubbles all have some common functionality, such as checking when it hits the bottom and movement along the Y axis. The object data structure is perfect for this. The question is, what should go into the Bubble class and what should be left out?

Global vs. local variables

At the very top of your pseudocode you can declare variables that will be needed throughout the entire program. These are called **global variables** as all functions/modules can access these variables within your program. The declaration of the bubbles array is an example in our pseudocode as it is needed throughout the whole program. **Local variables**, on the other hand, are declared within a particular function or module and cannot be accessed outside that function or module. You can pass local variables to other functions/modules as parameters. A good programming principle is to keep as many variables local as possible. This minimises inadvertent changes to the variable in other parts of your program. Some variables will need to be global, but they should be kept to a minimum.

Algorithms for objects and classes

In our bubbles program, the bubbleX and bubbleY values are local to the mouse-pressed function and are passed to the Bubble class. We can use **parameters** to pass the desired data into an object by assigning the constructor variables to the values being passed in. To make the data passed into a class unique to an instantiated object, the word 'this' is used to represent 'this' object. In the next example, we will see that when the mouse is pressed, the data is passed from an event listener to the class constructor.

Now start to develop your ideas such as a Bubble class into an algorithm. Remember that the class is simply a template or blueprint for the object that will be made from that class, and should be made as independent as possible from the main program so that it can be re-used easily. Note that the pseudocode below follows convention by using a capital letter for a class name. The pseudocode shown here is one of many possible approaches. You will improve on your first attempt by testing and refining as you go along. In this example, it is important that a bubble has an X and Y position, that it is displayed and that it moves.

ACTIVITY 1.3.12

Research the difference between a class and an object. List the advantages of using objects in your programming.



Develop

Define the data structures used

DECLARE bubbles array

//to store bubbles, example of a global variable

DECLARE Bubble class

CONSTRUCTOR (bubbleX,bubbleY,bubbleR)

this.bubbleX = bubbleX

this.bubbleY = bubbleY

this.bubbleR = bubbleR

//this is defining the class structure or layout

//constructor handles class data passed into it

//X position of the bubble, local variables

//Y position of the bubble

//radius of the bubble

//methods

showBubble

outline white

fill in bubble

OUTPUT bubble using bubbleX, bubbleY, bubbleR

moveBubble

//method also called a function

bubbleY = bubbleY + 1;

Create an event listener

Create a mouse-pressed event listener that creates a new bubble object when mouse is clicked.

function listen for mouse pressed

DECLARE newBubble with (mouseX,mouseY,bubbleR) //position at mouseX and Y

APPEND newBubble to bubbles array //store new bubble to the end of bubbles array

Show and move each bubble object in the array

LOOP until end of program

FOR index = 0 to length of bubbles array

 bubble.show the object in indexed array

 bubble.move the object in indexed array

Generate the code

You will move between generating and developing phases of the problem-solving process as your program evolves. The code generated in p5.js up to now is as follows:



```
let bubbles = []; //array to store new bubbles
let index = 0; //index pointer to array elements

//this function runs once at the start of the program
function setup() {
  createCanvas(400, 400) //sets up a 400x400 pixel canvas
}

//this function loops continuously throughout the program
function draw() {
  background(0); //set background colour to black
  for (let i = 0; i < bubbles.length; i++) {
    bubbles[i].show();
    bubbles[i].move();
  }
}

//this function listens for a mouse-pressed event and increments the
// array index
function mousePressed() {
  let newBubble = new Bubble(mouseX, mouseY, 25); //create an instance of bubble
  //class. Pass the mouse x and y and radius as parameters
  bubbles.push(newBubble);
}

//this function is the class constructor for bubble
class Bubble {
  constructor(x, y, r) {
    this.x = x;
    this.y = y;
    this.r = r;
  }
}
```

```

show() {
    stroke(255);
    fill(20);
    ellipse(this.x, this.y, this.r * 2);
}

move() {
    this.y = this.y + 1;
}
}

```

ACTIVITY 1.3.13

What does the bubble currently do when the mouse is clicked on the screen? Generate the code and see if you can develop the pseudocode for the bouncing of the bubble without reading ahead. Use tracing tools to watch what is happening as the bubble drops.



ACTIVITY 1.3.14

Draw up a trace table and trace the above algorithm to see if the logic will make the bubble bounce back up. Make up a starting Y and height of your screen for testing purposes. An example of a trace table might be the one shown here.

Something is wrong. Now it gets caught in a loop on the bottom of the screen.

HEIGHT OF SCREEN	BUBBLEY
5	1
5	2
5	3
5	4
5	5
5	6
5	5
5	6

ACTIVITY 1.3.15

Draw up a trace table and trace the revised bubble algorithm to see if the logic will make the bubble bounce back up. An example of a trace table might be the one shown here.

Something is wrong. Now it goes back up, but won't stop.

HEIGHT OF SCREEN	BUBBLEY	SPEED
5	1	1
5	2	1
5	3	1
5	4	1
5	5	1
5	6	-1
5	5	-1
5	4	-1
5	3	-1
5	2	-1
5	1	-1
5	0	-1
5	-1	-1
5	-2	-1

Evaluate the solution

We still need to refine the bouncing of the bubble.

Currently the bubble drops down but does not come back up in a bouncing motion. We know this because in bubble.move the pseudocode is 'bubbleY = bubbleY + 1'. As this loops around, the Y position is being incremented, making the bubble animate vertically down the screen. We need to detect when the bubble has hit the bottom so it can bounce back up. We need a selection that will check when bubbleY has gone past the height of the screen. We use a comparison operator in this case. There are many comparison operators we can use in a selection, such as <, >, <=, >= and != (is not equal to). Our logic now would be:

LOOP

IF bubbleY > height of screen

 bubbleY = bubbleY - 1

ELSE

 bubbleY = bubbleY + 1

Will the above logic work?

In the pseudocode above we have been successful in turning the bubble back up, but it only lasts until the bubbleY position is less than the height. We need a way to permanently turn the bubble around until it gets to the top and then drops again until no motion. The answer is with the variable speed.

DECLARE speed = 1

LOOP

 bubbleY = bubbleY + speed

IF bubbleY > height of screen

 speed = -speed

END IF

END LOOP

Recommending improvements

A clue to solving this comes from our basic mathematics knowledge. When we multiply something by -1 , the number changes to its opposite so, if positive, it would become negative, and vice versa. We need to combine logic conditions here with what are known as logic operators. Logic operators we can use are AND, OR and NOT. Now let's look at the logic of our pseudocode, knowing we can use the logic operator to combine conditions.

DECLARE speed = 1

LOOP

bubbleY = bubbleY + speed

IF bubbleY > height of screen **OR** bubbleY < 0

 speed = speed * -1

To make the bubbleSpeed part of an object instance, you would pass the speed data into the class constructor, just as we did with X, Y and R. The Bubble class structure would now look like this:

ACTIVITY 1.3.16

Draw up a trace table and trace the above algorithm to see if the logic will make the bubble bounce back and forth. Has this solved our original brief?

```
//this function is the class structure for bubble
class Bubble {
  //the constructor handles the class data
  constructor(bubbleX, bubbleY, bubbleR, bubbleSpeed) {
    this.bubbleX = bubbleX;
    this.bubbleY = bubbleY;
    this.bubbleR = bubbleR;
    this.bubbleSpeed = bubbleSpeed;
  }

  //this method displays the bubble
  show() {
    stroke(255);
    fill(20);
    ellipse(this.bubbleX, this.bubbleY, this.bubbleR * 2);
  }

  //this method moves the bubble
  move() {
    if (this.bubbleY > height || this.bubbleY < 0) { //hits top/bottom
                                                    //change direction
      this.bubbleSpeed = this.bubbleSpeed * -1;
    }
    this.bubbleY = this.bubbleY + this.bubbleSpeed; //move bubble
  }
}
```

You should also notice the variable names are now being made very clear and meaningful.

You would also need to declare speed at the top of your code:

```
let bubbleSpeed = 3; //speed of bubble
```

You would also need to add the speed parameter to the new bubble when it is instantiated.

```
let newBubble = new Bubble(mouseX, mouseY, 25, bubbleSpeed);
```

ACTIVITY 1.3.17

Discuss why the speed parameter needs to be added to the instantiation of a new bubble. Try coding the bubble without this parameter and observe the results.

Generate and evaluate your code so far. Observe what happens when you click the screen.

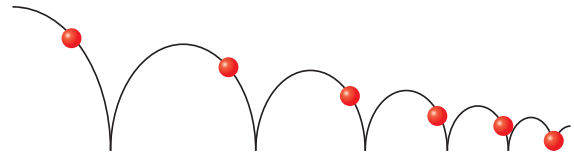


Figure 1.3.20 Gravity on bouncing ball

Making the bubbles bounce

Our final part of this task is to make the bubbles bounce, rather than simply go back and forward continuously. Exploring this further, we look into the world of physics and mathematics and consider what happens when we bounce a ball as shown in figure 1.3.20.

For the purposes of this example, we will use a gravity effect of $\frac{3}{4}$ or 0.75 on the bubble, so that each time the bubble bounces, it comes back up 0.75 of its previous height. Because we are working from the top of the screen when working with Y direction, we need to add 0.25 times the previous bounce height each time. To do this, we need a variable to store the previous height and gravity data. Observe now how much our pseudocode has developed over the journey of this project.

Pseudocode so far

```

DECLARE index                                //pointer for the bubble array
DECLARE newBubble                             //store a new bubble when mouse clicks screen
DECLARE bubbles array                        //to store bubbles
DECLARE bubbleSpeed = 3                      //pixels the bubble moves each loop
DECLARE gravity = 0.25                      //fraction of previous bounceheight from top
CONSTRUCTOR (bubbleX,bubbleY,bubbleR, bubbleSpeed, gravity)
    this.bubbleX = bubble                    //X position of the bubble
    this.bubbleY = bubbleY                  //Y position of the bubble
    this.bubbleR = bubbleR                  //radius of the bubble
    this.bounceHeight = bubbleY             //store the bounce height
    this.gravity = gravity

//methods
showBubble
    outline white
    fill in bubble
    OUTPUT bubble using bubbleX, bubbleY, bubbleR

moveBubble
    IF bubbleY > height OR bubbleY < bounceHeight THEN    //detect top or bottom
        bubbleSpeed = bubbleSpeed * -1                    //reverse direction
    END IF
    bubbleY = bubbleY + bubbleSpeed                        //move bubble

```

p5.js code so far

The code generated so far in p5.js is:

```
let bubbles = [];           //array to store new bubbles
let index = 0;              //index pointer to array elements
let bubbleSpeed = 3;        //speed of bubble
let gravity = 0.25;

//this function runs once at the start of the program
function setup() {
  createCanvas(400, 400) //sets up a 400x400 pixel canvas
}

//this function loops continuously throughout the program
function draw() {
  background(0);           //set background colour to black
  for (let i = 0; i < bubbles.length; i++) {
    bubbles[i].showBubble();
    bubbles[i].moveBubble();
  }
}

//this function listens for a mouse-pressed event and increments the array index
function mousePressed() {
  let newBubble = new Bubble(mouseX, mouseY, 25, bubbleSpeed, gravity);
  bubbles.push(newBubble); //appends the new bubble to end of the array
}

//this function is the class structure for bubble
class Bubble {
  //the constructor handles the class data
  constructor(bubbleX, bubbleY, bubbleR, bubbleSpeed, gravity) {
    this.bubbleX = bubbleX;
    this.bubbleY = bubbleY;
    this.bubbleR = bubbleR;
    this.bounceHeight = bubbleY; //stores the height of the bubble
    this.bubbleSpeed = bubbleSpeed;
    this.gravity = gravity;
  }

  //this method displays the bubble
  showBubble() {
    stroke(255);
    fill(20);
    ellipse(this.bubbleX, this.bubbleY, this.bubbleR * 2);
  }

  //this method moves the bubble
  moveBubble() {
    if (this.bubbleY > height || this.bubbleY < this.bounceHeight) {
      //if it hits the top or bottom change direction
      this.bubbleSpeed = this.bubbleSpeed * -1;
    }
  }
}
```

```

        this.bubbleY = this.bubbleY + this.bubbleSpeed; //moves bubble
    }
}

```

Adding gravity

Up until now we have been using comparison and logical operators. We have also been using another type of operator known as the arithmetic operator. These are $+$, $-$, $*$, $/$, which are for addition, subtraction, multiplication and division. There are others that we will learn as we continue.

At this stage, gravity has not yet been applied. We need to consider the logic and develop our idea in pseudocode. Below is a solution to the 'move bubble' method from our Bubble class when we apply gravity.

```

IF bubbleY > height or bubbleY < bounceHeight THEN //detect top or bottom
    bubbleSpeed = bubbleSpeed * -1 //reverse direction
END IF
bubbleY = bubbleY + bubbleSpeed //move bubble
IF bubbleY > height THEN //detect bottom
    bounceHeight = bounceHeight + bounceHeight * gravity
END IF

```

ACTIVITY 1.3.18

Update your code to match the example above. Observe, discuss and explore your ideas, develop the pseudocode and generate and evaluate a solution.

ACTIVITY 1.3.19

- 1 Run your final solution after adding the gravity.
- 2 Explore gravity in terms of its effect on the bubble as it moves up and down.
- 3 Develop ideas to add to the existing solution (above) where gravity has been applied to the bounce height but not the speed of the bubble.
- 4 You could also develop ideas for a change in shape of the bubble when it hits the ground and a swirling motion up and down that simulates bubbles.

TEST YOUR KNOWLEDGE

- 1 Explain the difference between compiled instructions and interpreted instructions.
- 2 What is the purpose of a computer variable?
- 3 What type of variables are there?
- 4 Explain the different data structures used in computer programming.
- 5 Develop an IPO chart based on a user entering the minimum temperature and the maximum temperature for the day, to display the average temperature for the day.
- 6 Develop an algorithm using pseudocode for question 5.
- 7 Perform a desk check on question 6.
- 8 Develop an algorithm for a program that will accept the user's name and output a friendly message: "Hello <name>, how are you?".
- 9 What are the steps involved in a desk check?
- 10 Explain why it is important to perform a desk check on the algorithm idea.
- 11 Explain the purpose of code indentation.
- 12 List the control structures used in programming.
- 13 Explain the difference between a syntax error and a logic error.
- 14 Explain what an event listener is and what it is used for.
- 15 Explain the difference between a local and a global variable.
- 16 Explain the benefit of having a class constructor in an object.
- 17 Why is modularisation so important for structured programming?



APPLY YOUR KNOWLEDGE

- 1 Develop an algorithm for a simple calculator that has only the four basic operators +, −, *, /, and can only accept two numbers at a time. When the user hits the = button the answer is displayed.
- 2 Design an algorithm that will receive the weight of a parcel and determine the delivery charge for that parcel. Use the following table:

PARCEL WEIGHT (KG)	COST PER KILOGRAM
<2.5kg	\$3.50
2.5–5 kg	\$2.85
>5kg	\$2.45

- 3 Create a class for a car object with properties and methods, then display those properties on the screen.
- 4 Read through the Euclid's tiles solution and explore, develop, generate and evaluate your own solution using circled tiles.
- 5 Develop an algorithm and generate the code that will prompt for the measurements for the diameter of a circle, and calculate and display the area and circumference of that circle. Your program should exit if 9999 is entered.

1.4

PROGRAMMED SOLUTIONS

The aim of this section is to combine the explore, develop, generate and evaluate phases of the Digital Solutions problem-solving process to produce programmed solutions. Section 1.1 introduced scenarios that unpacked concepts about the exploration of digital solutions. Section 1.2 explored and developed ideas about user interfaces and section 1.3 looked further at the development of ideas through the use of structured programming principles, algorithms and the use of coding structures when generating solutions. This section will focus on generating and evaluating components of programmed solutions and applying concepts explored in previous sections.

Software frameworks and libraries

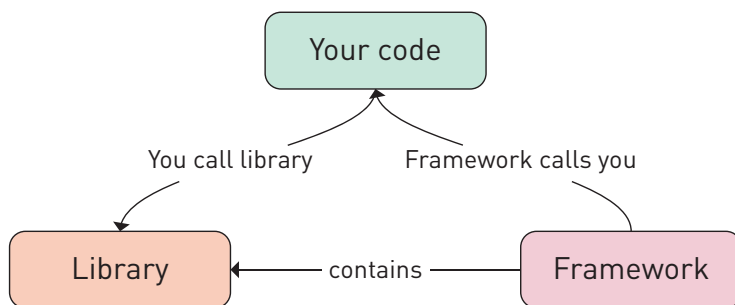


Figure 1.4.1 Software frameworks and libraries

Before diving into the generated solutions, it is worth understanding what software frameworks and libraries are. They are commonplace in software development, making it easier and faster to code and, often, extending capabilities.

The key difference between a library and a framework is ‘inversion of control’. When you call a method from a library, you are in control. But with a framework, the control is inverted: the framework calls you. In section 1.3, the

walk-through examples used a JavaScript library called p5.js. This gave the ability to control the way the code was constructed and to mix it with native JavaScript. Examples of frameworks are Microsoft’s .NET, which is a software development framework; Ionic, a mobile app development framework; Django, a web development framework; and front-end web development frameworks such as Bootstrap and JQuery. Frameworks follow a strict architecture.

Standards

There are some important precautions to take before using external libraries and frameworks. It is important to explore the developer’s adherence to standards because it is standards that determine longevity and global support. Your digital technology context will dictate where you look for standards. If it is a web application, then web standards (HTML, CSS and JavaScript) will apply.

The World Wide Web consortium (W3C) of industry and governments is a member of the International Organization for Standardization (ISO). Other members of ISO are ECMA International and Standards Australia (SA), dedicated to the global standardisation of information and communication technology. These organisations offer a good platform to explore standards in whatever language you decide to use.

ACTIVITY 1.4.1

Explore who sets the standards for some popular programming languages by going to the International Organization for Standardization website and typing the name of the language into the search engine at the top of the page. Follow links to explore the standards and who sets them.



1.4.1 Standards
1.4.2 International standards
1.4.3 Communications and consumer electronic standards
1.4.4 Standards Australia

A software stack is a group of programs that work in tandem to produce a result or achieve a common goal. A full-stack developer is someone who is familiar with all layers in computer software development. These developers aren't experts at everything; they have the functional knowledge and ability to take a concept and turn it into a finished product.



Web context: Rusty and the Ezibeats prototype programmed solution

Earlier sections explored and developed ideas in response to a client brief for Rusty and the Ezibeats' prototype web presence. Let's now generate the final prototype using HTML, CSS and JavaScript. In this example, we use Visual Studio Code (VSC) as our development environment. We will also use Chrome debugging tool extensions within VSC. We will explore adherence to standards in frameworks, pre-developed classes and code libraries.

Setting up the development environment

Take time to set up the development environment correctly. This will help minimise frustration and development problems as you start to generate. The important steps in setting up are as follows:

SETTING UP YOUR PROJECT

- 1 Create the appropriate folders for the project on your device.
- 2 Collect the digital assets needed and store them in appropriate folders for the project.
- 3 Download relevant libraries into appropriate folders for the project.
- 4 Install required extensions to the development environment.
- 5 Configure the development environment to use the project folders.

Backup

Copy the project folder and its contents to a backup location to prevent loss of work at the end of each session. This practice safeguards your work and provides version control. This is particularly helpful after major changes have been added to your program. If something goes wrong with the code, you can always roll back to the previous version you have backed up.

Generating user interfaces

Using third-party libraries and frameworks often speeds up the time spent in development. The pre-developed CSS library called w3.css, from w3schools.com, contains many common user interface styles. Most external libraries and frameworks are continually updated to comply with new standards. Unless they use meaningful naming standards, however, it can be difficult to understand and use the styles they provide. Developers must decide whether to use pre-defined styles or create their own. There are advantages and disadvantages in both approaches.

In a web application, the user interface consists of the HTML document, which provides a structure to the interface, and a CSS styles file, which controls the appearance and layout of each web page.

CSS

Cascading style sheets (CSS) define a hierarchy of styles for elements on a web page. The cascading feature means that default styles defined for outermost parts of the website filter down to the innermost elements. Redefining styles for inner elements overrides the default style set in the container for the element. Style libraries typically use classes to group styles for specific types of elements in a web page. Multiple classes can be combined to create a unique style for each web element. Attaching the appropriate style sheet will automatically format the web page to these predefined styles. If a more specific style is required, additional style rules can be defined to supplement or override the styles provided by the library style sheet. CSS can be used to change colours, fonts, padding, margins, borders and many other aspects of the appearance of a web page. They are critical to generating an engaging web application user interface with good useability features.

HTML

The HTML file defines the structure of the user interface of the web application. It contains two main parts: a head section and a body section.

The head section of the HTML page is not shown in a web browser. It contains meta-information about the page, such as a page title and links to other external library files such as CSS and JavaScript.

Table 1.4.1 The head section of the HTML document that defines the user interface for the Ezibeats website

HTML	COMMENT
<code><head></code>	(displayed in browser tab)
<code><title>Rusty and the Ezibeats</title></code>	Title of the page
<code><meta charset="UTF-8"></code>	Specifies what character set your website is encoded with
<code><meta name="viewport" content="width=device-width, initial-scale=1"></code>	Sets the screen size to full size of the device
<code><link rel="stylesheet" href="w3.css"></code>	Links a CSS stylesheet to the HTML page
<code><link rel="stylesheet" href="https://fonts.googleapis.com/css?family=Lato"></code>	Links Google CSS font library
<code><link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/4.7.0/css/font-awesome.min.css"></code>	Links an icons library
<code><script type="text/javascript" src="scripts.js" defer></script></code>	Links the JavaScript file
<code></head></code>	

Fonts and icons

The standard fonts and icons available to a web page are very limited. Most developers will use the head section of the HTML page to link to external font libraries and icon libraries to increase the range of fonts and icons available. Table 1.4.1 shows how external fonts and icons are linked to the Ezibeats website.

1 Fonts

The font used on the Ezibeats site is called LATO and comes from the external CSS Google Fonts library:

```
<link rel="stylesheet" href="https://fonts.googleapis.com/
css?family=Lato">
```

Once the external Google library has been linked, the LATO font can be used in a style rule, as follows:

```
body {
    font-family: "Lato", sans-serif;
    font-size: 15px;
    line-height: 1.5
}
```

2 Icons

Icons such as the down arrow on the drop-down box and the magnifying glass on the search bar are linked from the Font Awesome CDN website, a free and public CSS library. Once this link is in place, default styles can be overridden by adding the class name to the HTML tag. In this case, the class name is 'fa fa-caret-down' and looks like this:

```
<i class="fa fa-caret-down"></i>
```

On the mobile-size page, the tag is:

```
<i class="fa fa-bars"></i>
```

The HTML body section contains the content, including the navigation bar, web page banner, footer and multimedia content.

Navigation bar

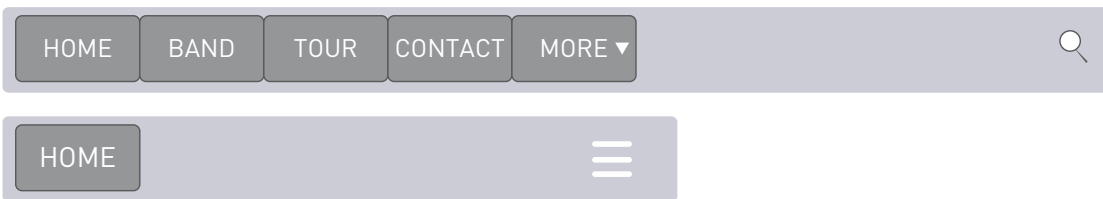


Figure 1.4.2 Responsive navigation bars

The navigation bar content is defined by opening and closing '`<nav>`' and '`</nav>`' tags in the HTML file. Without styling, the navigation bar content would appear as an underlined list of hyperlinks. By using the styles in the w3.css library, the navigation links are transformed into an attractive menu bar. Figure 1.4.2 shows how the style displays the navigation bar in wide format on a desktop view and condensed format in mobile view. Table 1.4.2 shows how the styles are applied using the class attribute of each HTML tag.

Table 1.4.2 HTML and CSS for navigation on Rusty and the Ezibeats website

HTML	CSS
<pre> <!-- Navbar --> <nav class="w3-top w3-hide-small"> <div class="w3-bar w3-black w3-card"> HOME BAND TOUR CONTACT <div class="w3-dropdown-hover"> <button class="w3-padding-large w3-button" title="More">MORE <i class="fa fa-caret-down"></i> </button> </div> </div> </nav> </pre>	Navbar positioning <pre> w3-top, w3-button { position: fixed; width: 100%; z-index: 1; } .w3-top { top: 0; } </pre>

<pre> <div class="w3-dropdown-content w3- bar-block w3-card-4"> Merchandise Extras Media </div> </div> <i class="fa fa-search"></i> </div> </nav> <!-- Navbar on small screens --> <nav class="w3-top w3-hide-medium w3-hide- large"> <div class="w3-bar w3-black w3-card"> HOME <i class="fa fa-bars"></i> <div id="dropDown" class="w3-bar-block w3-black w3-hide" style="margin- top:46px"> <div class="w3-bar w3-black w3-card"> BAND TOUR CONTACT MERCH </div> </div> </div> </nav> </pre>	Rollover on buttons. Changes the background to light grey and font to black using CSS rule <pre> .w3-button { border: none; display: inline-block; outline: 0; padding: 8px 16px; vertical-align: middle; overflow: hidden; text-decoration: none; color: inherit; background-color: inherit; text-align: center; cursor: pointer; white-space: nowrap } .w3-button:hover { color: #000!important; background-color: #ccc!important } </pre>

The navigation bar stays at the top of all pages because of the ‘position:fixed’ rule and in front of the content because of the ‘z-index:1’ rule. It spans across the entire page because of the ‘width:100%’ rule.

How does the navigation bar change when its screen size (viewport) is reduced? It does this with a **media query** and by hiding and unhiding elements on the page using CSS styles. A media query is like an IF-THEN selection event listener that continually listens for changes in viewport size. For example, ‘IF the viewport is larger than 993 px, then only display the large navigation element’; ‘IF the viewport size is less than 600 px, only display the small navigation element’. This particular template is responsive at three checkpoints (viewport sizes) – for desktop, tablet and mobile. The navigation bar, however, has only two HTML elements that are either hidden or displayed depending on the viewport size.

```

@media (min-width:993px) { .w3-hide-large {display: none!important } }
@media (max-width:992px) and (min-width:601px)
{ .w3-hide-medium { display: none!important } }
@media (max-width:600px) { .w3-hide-small { display: none!important } }

```

ACTIVITY 1.4.2

Explore the media query syntax above. Identify where the HTML classes are that match the media queries; interpret the code to hypothesise at what viewport width the hamburger menu appears. Try using media queries to change the background colour of a blank HTML page based on the browser size.

Figure 1.4.3 shows two different views of the same web page. Media queries have been used to select the appropriate menu for the viewport size.

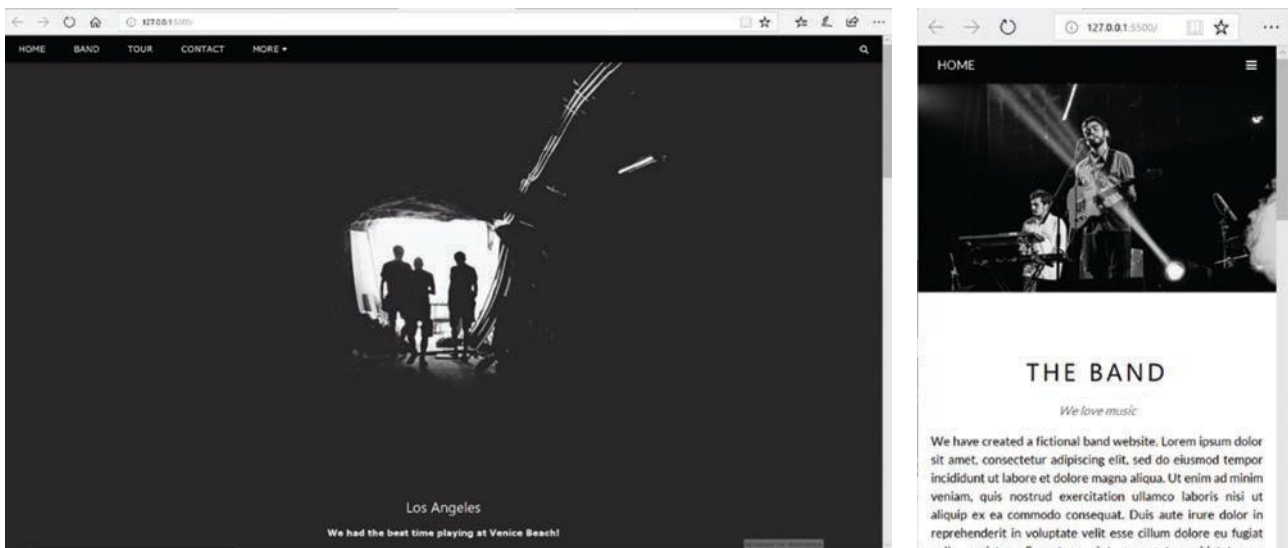


Figure 1.4.3 Home page with animated photos

You will notice in Table 1.4.2 that, while the HTML for the standard navigation bar uses hyperlinks, in the section for navigation on small screens there is a call to a JavaScript function when the hamburger icon is pressed. That is to show and hide the dropdown hamburger menu when the user presses it on small devices. The JavaScript code for this function is shown below.

```

// Used to toggle the menu on small screens when clicking on the menu button
function myFunction() {
  var x = document.getElementById("dropDown");
  if (x.className.indexOf("w3-show") == -1) { // checks if drop list is displayed,
                                                // -1 returns closed
    x.className += "w3-show"; //show drop list
  } else {
    x.className = x.className.replace("w3-show", ""); //hide drop list
  }
}

```



This works because the JavaScript can access CSS classes. 'w3-show' is a display box for the drop-down list. This is like an array or list data structure that can be searched. If the user presses the icon, the menu will appear because the selected option is not in the drop-down list. The `indexOf` returns `-1` if the user presses anything but the menu options. This algorithm works well because the menu will remain while the user presses menu options.

ACTIVITY 1.4.3

Below is the HTML for the drop-down menu in the large-size navigation bar. Several classes have been combined to create the appearance of the navigation bar. Inspect the `w3.css` file and identify the CSS rules used and how they contribute to the appearance of the navigation bar.

```
<div class="w3-dropdown-hover">
  <button class="w3-padding-large w3-button" title="More">MORE
    <i class="fa fa-caret-down"></i>
  </button>
  <div class="w3-dropdown-content w3-bar-block w3-card-4">
    <a href="#" class="w3-bar-item w3-button">Merchandise</a>
    <a href="#" class="w3-bar-item w3-button">Extras</a>
    <a href="#" class="w3-bar-item w3-button">Media</a>
  </div>
</div>
```

Home page layout and content

The elements on any HTML page are defined using HTML tags enclosed in angled brackets – for example, `<div> ... </div>`, `<a> ... </a>`. Each pair of tags represents a rectangular box on the web page that contains the web page content, such as images, text etc. The margin around the edge of the box, the border around the box and the padding within the box can be styled using CSS. In section 1.2 the wireframes for the user interface of the Ezibeats website showed the layout of boxes on the page. This approach to designing a web page by styling padding, margin and borders of rectangular boxes is known as the CSS box model.

In responsive design the size of each box is not fixed. It needs to vary at each breakpoint in the viewport used to display the web page. As shown in the code sample below, the `w3.css` style library uses maximum and minimum values for the width and height of web page content boxes to ensure that the boxes will automatically resize to fit the size of the viewport. Planning and testing a responsive design usually involves considering how each HTML element aligns and interacts with other elements at a variety of screen sizes. This can be a very time-consuming exercise. Using a style library that supports multiple viewports is an efficient approach to creating a responsive website.

```
.w3-content {
  max-width: 2000px;           /*that's a big screen*/
  margin: auto;                /*centre the page*/
  margin-top: 46px;           /*to bring it down below the navigation bar*/
}
```

HTML for the slideshow

The essential building blocks of HTML were designed decades ago for static web pages of basic text and an occasional image or hyperlink. Although the basic building blocks of HTML have not significantly changed, the features of web pages continue to evolve, with new trends emerging every few years. Slide shows are a popular feature of many home pages but there is no `<slideshow>` tag in HTML. Slideshow effects are achieved with simple image boxes and heading tags controlled by JavaScript, as shown in the code sample below. Each photo is loaded in its own division or box and then hidden until it is ready to be displayed and animated in the slideshow.



1.4.6
Introduction
to the CSS
basic box
model
1.4.7 w3.css

```

<!-- Automatic Slideshow Images -->
<div class="mySlides w3-display-container w3-center">
  
  <div class="w3-display-bottommiddle w3-container w3-text-white w3-
    padding-32 w3-hide-small">
    <h3>Los Angeles</h3>
    <p>
      <b>We had the best time playing at Venice Beach!</b>
    </p>
  </div>
</div>
<div class="mySlides w3-display-container w3-center">
  
  <div class="w3-display-bottommiddle w3-container w3-text-white w3-
    padding-32 w3-hide-small">
    <h3>New York</h3>
    <p>
      <b>The atmosphere in New York is lorem ipsum.</b>
    </p>
  </div>
</div>
<div class="mySlides w3-display-container w3-center">
  
  <div class="w3-display-bottommiddle w3-container w3-text-white w3-
    padding-32 w3-hide-small">
    <h3>Chicago</h3>
    <p>
      <b>Thank you, Chicago - A night we won't forget.</b>
    </p>
  </div>
</div>

```

Javascript for the slideshow

```

// Automatic Slideshow - change image every four seconds
var myIndex = 0; //starts at the first photo in the array when page loads
carousel(); //calls the function carousel automatically when page loads

function carousel() {
  var i;
  var x = document.getElementsByClassName("mySlides");
  for (i = 0; i < x.length; i++) { //loop through the array and hide all photos
    x[i].style.display = "none";
  }
  myIndex++;
  if (myIndex > x.length) { //check if going past the last photo
    myIndex = 1 //goes back to the first photo when reaches the end of
    //the array
  }
  x[myIndex - 1].style.display = "block"; //displays each photo for 4 seconds
  setTimeout(carousel, 4000); //executes carousel, after waiting 4 seconds
}

```

The images reduce in size for a mobile device due to the CSS style contained within the class 'w3-center'. It has a rule, 'width:auto;', which automatically sizes the image to the browser.

We will focus on an animation effect on the ticket purchase page. The other pages use the same principles applied and learnt above. When the 'buy tickets' button is pressed, a purchase ticket form animates down from the top of the screen using a CSS animation, as shown in figure 1.4.4.

The HTML for the purchase ticket button is:

```
<button class="w3-button
w3-black w3-margin-
bottom" onclick="document.
getElementById('ticketModal').
style.display='block'">Buy
Tickets</button>
```

The form element 'ticketModal' has a class named 'w3-animate-top'. The CSS3 animation syntax is shown below. Since this class involves animation, the rule, '@keyframes', defines the start and finish states. The @keyframe rule is activated using its name 'animatetop'. The animation property defines which keyframe rule to use and the duration of the animation.

```
.w3-animate-top {
  position: relative;
  animation: animatetop 0.4s;
}
@keyframes animatetop {
  from {
    top: -300px;
    opacity: 0;
  }
  to {
    top: 0;
    opacity: 1;
  }
}
```

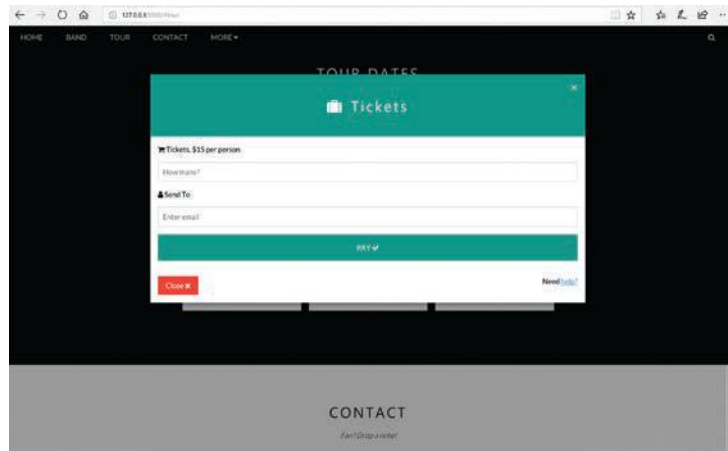


Figure 1.4.4 Animated drop-down

ACTIVITY 1.4.4

Explore some other styles of animation from the Mozilla website and create a blank HTML page with some fa fa icons inserted from the previous activity. Attempt to apply some animations to your icons.



1.4.8
Using css
animations

Evaluation checklist

The final user interface should be thoroughly tested and evaluated. Many online tools can be used to test the effectiveness and utility of web interfaces. It is worth exploring your own digital technology context to find some tools and steps to automate some of the testing and evaluation of a completed user interface. Useability and user experience evaluation should be informed by collecting feedback from a sample target audience of users. Table 1.4.3 is an example of a traditional tool used for collecting user and developer feedback. Most users, however, do not have time to fill in lengthy, complicated forms. A rating style is often used, with language that a user can understand.

Table 1.4.3 Browser testing

BROWSER TESTING CHECKLIST	
Project name:	<required>
Project manager name:	<required>
Project description:	<required>
BROWSER TEST PLAN	
Which browser versions are being used by your target audience?	
Can you ensure the availability of all these browser versions?	
Which browser/hardware is needed for conducting these browser tests?	
CONTENT	Yes/no
Are all the images available across browsers?	
Are used fonts working for every browser version?	
Are all audio files available for every browser?	
Can the content be consumed on every browser version?	
LAYOUT	Yes/no
Is the layout consistent?	
Are the layout elements behaving as expected?	
Is the (responsive) browser layout working in all browser versions?	
Are animations and effects running smoothly and as expected?	
FUNCTIONALITY	Yes/no
Are all features usable in the targeted browser versions?	
Are all available integrations and third-party services running as expected?	
Are all forms and other dynamic interactions working across browsers?	
Are all interactions working equally well with touch input (e.g. from a mobile device) and mouse input (e.g. from a desktop computer?)	

ACTIVITY 1.4.5

In section 1.2 the useability principles were explored, including accessibility, effectiveness, safety, utility and learnability. These are all measurable qualities of the final digital solution.

- 1 Generate a useability evaluation tool using the layout of the table above as a guide.
- 2 Explore more online tools that might assist with useability testing.



1.4.9
PageSpeed
insights

CASE STUDY**ASCII ALIENS**

In section 1.1, we explored the ASCII Aliens game using a mind map. This identified a 'binary ship', which was used to fire 'binary bullets' (bits) – either a 0, by pressing the 0 key, or a 1, by pressing the 1 key – at the ASCII Aliens. We also identified a sequence of events occurring in the game. In section 1.2, some user interface ideas were mocked up in a wireframe. In this section we will focus further on the development of ideas by adding more technical detail, using diagrams and pseudocode, which will need to be tested using desk checking, and then start generating and evaluating code.



Develop data structures, procedures (functions), hierarchy and pseudocode

In previous sections we explored simple and complex data structures. Here we can apply that knowledge to decide how best to organise the data we will need for the game. First, let's look at a tool to represent the game objects. You can use a class diagram to illustrate a group of objects and list their properties (data) and methods (functions). The class diagram for the ASCII Aliens game is shown in figure 1.4.5.

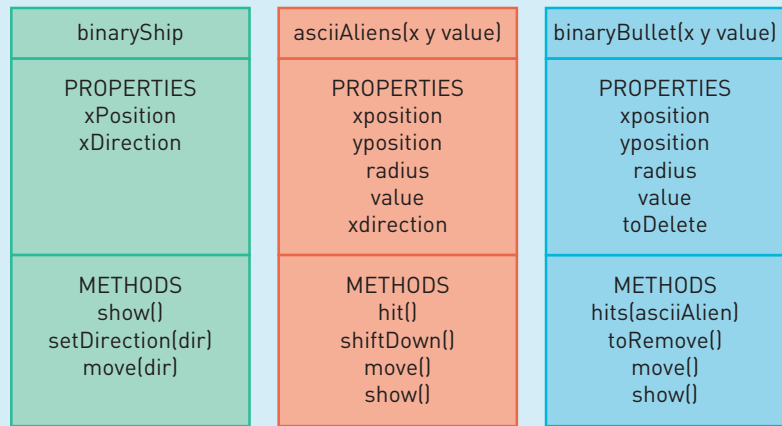


Figure 1.4.5 Class diagrams

ACTIVITY 1.4.6

Explore some real-life objects and, in groups, develop class diagrams to represent your ideas

An array or list is used to store a group of related items. In ASCII Aliens an array should be used if there are many of the same data type being used often throughout the game. In this case, the 'binaryBullet' objects are a great example of the way an array can help keep track of objects of the same type. The size of the array changes constantly because a new bullet is added when fired and removed if it hits the Alien or is off the screen. An array is also good for storing the eight Alien objects rather than have individual variables for each. The other use for an array is for storing the eight-BIT binary number for the secret symbol, as we need to use each bit regularly. The tables below are used to illustrate how the computer memory could store and manage the ASCII Aliens game data.

binaryBullets

0	1	2	3	4
binaryBullet object	binaryBullet object	binaryBullet object	binaryBullet object	binaryBullet object

The binaryBullets array has no fixed size.

asciiAliens

0	1	2	3	4	5	6	7
AsciiAlien object	AsciiAlien object	AsciiAlien object	AsciiAlien object	AsciiAlien object	AsciiAlien object	AsciiAlien object	AsciiAlien object

binaryArray

0	1	2	3	4	5	6	7
0	0	1	1	0	1	1	0

The other data structures to be used are variables, which we will examine more when developing the pseudocode.

The data and functions relating to the three objects binaryShip, asciiAlien and binaryBullet can be developed further, using pseudocode:

```
//binaryShip()
//properties
x position = middle of the screen    //position ship
horizontal direction = 0              //direction stopped = 0, right = 1, left = -1
//methods
```

```

show()                //method to show the ship
    MAKE colourfill red
    DISPLAY rectangle at x = x, y = 20 up from bottom, w = 20 h = 60
setDir(dir)           //gets direction from left and right arrows from keyboard
    horizontal direction = dir    //passes the keyboard value to the binaryShip property xdir
move(dir)             //moves ship based on keyboard value
    move 5 pixels * horizontal direction    //negative direction = left, postive direction = right
END

```

ACTIVITY 1.4.7

Evaluate the binary ship pseudocode by developing a trace table and then step through one press of an arrow key in either direction.

```

//AsciiAlien(x, y, value)
//x, y are position co-ords for alien passed in from game. Value is the BIT value of secret symbol
// properties
x position = x
y position = y
radius = 20                //size of alien radius
bit value = value          //stores a BIT value of the secret symbol
horizontal direction = 1    //direction starts right
//methods
hit()
    radius = 0              //make alien disappear
    DISPLAY bit value at x position and y position    //show BIT value of alien
shiftDown()
// this function moves the aliens down and in opposite direction when they hit the edge
    horizontal direction accelerate by .1 and change direction
    Add radius to y position    //lowers alien down by size of radius. If hit alien doesn't
                                //move down

move()
//this function moves aliens in x direction when called, controlled by keyboard
    Add horizontal direction to x position
show()
    MAKE fill colour red
    DRAW ellipse at x position, y position using radius x 2 as height and width
    OUTPUT BIT value at x position, y position
END

//BinaryBullet(x, y, value)
//PROPERTIES
x position = x
y position = y
radius = 8
bit value = value
delete flag = false        //flag to indicate bullet needs removing
//METHODS
show()                      //this displays the bullet and bit
    MAKE FILL colour blue
    DISPLAY ellipse at x position, y position using radius x 2 as width and height    //ellipse
                                                for bullet

```

```

    DISPLAY BIT value on bullet //the bit is display on bullet

hits(asciiAlien)
    //if distance from centre of bullet to center of alien is
    //greater than radius of bullet + radius of alien then a hit
    DECLARE distance = distance between alien and bullet
    IF distance < bullet radius + alien radius THEN
        return true //a hit and true sent back
    ELSE
        return false //a miss and false returned back
    ENDIF

toRemove ()
    //removes the bullet
    delete flag = true

move ()
    move y position down 5 pixels
END

```

In the pseudocode above, the `asciiAliens` and `binaryBullets` have parameters written in the form `asciiAliens(x,y,value)` and `binaryBullets(x,y,value)`. Parameters are inputs to a function or method that will be used as local variables within the function. The alien and bullet objects get their x, y positions and value (binary digit) from the calling program. Parameters are a good way to pass data within your code.

With data structure ideas developed, it is time to develop ideas about the sequence and procedures in a hierarchy chart, as shown in figure 1.4.6.

This structure then guides the development of the pseudocode, which sets out the algorithms that will then be generated into code. Over the next few pages some interesting, effective and efficient algorithms have been developed and desk checked using processes explained in previous sections.

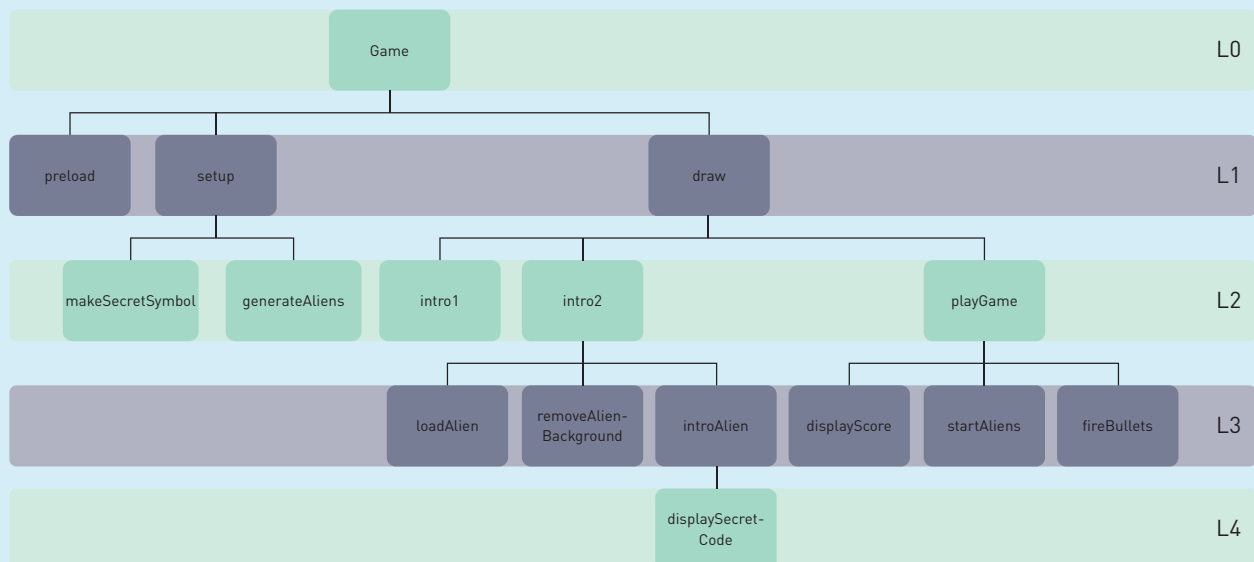


Figure 1.4.6 Hierarchy chart, ASCII Aliens

Level 0 pseudocode

```
//declare global variables
DECLARE class binaryShip
DECLARE backgroundImage
DECLARE asciiAliens array = [] //array to store aliens
DECLARE binaryBullets = [] //array to store bullets
DECLARE exitIntro1 = false //flag to indicate when intro1 is over
DECLARE secretSymbol // store the random secret symbol
DECLARE ascii //stores the ascii code for the secret symbol
DECLARE binaryArray = [] //stores the secret binary number in an array
DECLARE binaryNumber = " //stores the secret binary number as a string
DECLARE alienCount = 0 //used to move alien
DECLARE beam = 0 //used to beam down secret symbol
DECLARE ammo = 0 //bullets used
DECLARE score = 0 // 1 hit = 5 points, 1 bullet = -1, aliens down = -2
DECLARE kills = 0 //number of aliens eliminated
```

Level 1 pseudocode

```
//preload assets – this will run only once before game starts
Load backgroundImage
Load alienImage

//setup runs once before game starts. Setup a game canvas, generate a random character, create an array
//of 8 alien instances
Setup resolution of game
Setup canvas size 800 x 600
MakeSecretSymbol //generate a random character into binary
generateAliens //array of 8 aliens

//draw loops until game finished. Loads the background image, calls introduction then runs the game
LOAD backgroundImage at x = 0 y = 0 w = 800 h = 500
IF exitIntro flag is not true THEN
    CALL intro1 //displays game instructions then waits for key to be pressed
ELSEIF exitIntro2 flag not true THEN
    CALL intro2 //animates alien beamed down with secret symbol
ELSE
    playGame //run the game until finished
ENDIF
ENDIF
```

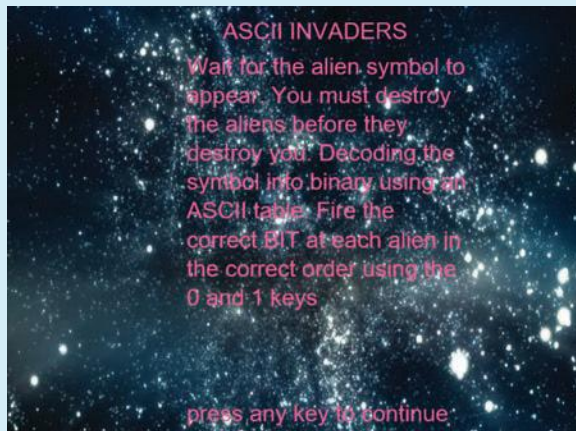
Level 2 pseudocode

```
//makeSecretSymbol
//is called only once in setup to create a secret symbol and convert it to an ascii code and then into a
//binary number
binaryShip = new BinaryShip //new instance of BinaryShip class
secretSymbol = random number from 65 to 90
ascii = the ascii code for secretSymbol //returns the ascii code for the number
binaryNumber = ascii to binary //returns the binary number
```

```
//generateAliens  fills an array of 8 alien objects each positioned along the x direction
FOR i = 0 to i < 8  i++ DO
  asciiAliens[i] = new AsciiAlien OBJECT AT X = i * 50 + 50, Y= 100
  asciiAliens[i].value = binaryArray[i]    // assigns a bit to an alien
ENDFOR
```

Concept drawings are from wireframes in section 1.2 to add further detail to the development of ideas.

```
//intro1
```



```
SET text size 32
```

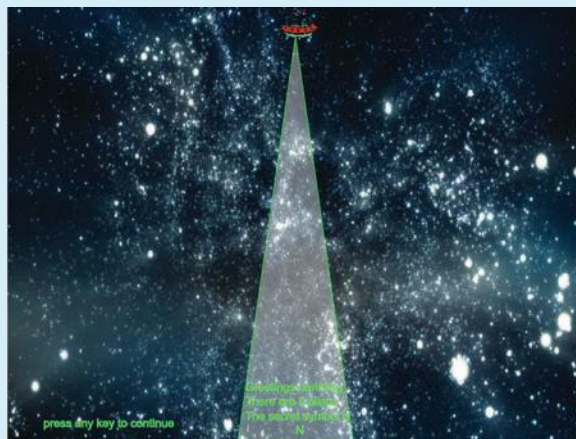
```
DISPLAY "ASCII INVADERS" at x = 300 y= 50
```

```
SET text colour pink
```

```
DISPLAY "Wait for the alien symbol to appear. You must
destroy the aliens before they destroy you. Decoding the
symbol into binary using an ASCII table. Fire the correct
BIT at each alien in the correct order using the 0 and 1
keys" at x=250 y=70 w=400 h=400
```

```
DISPLAY "press any key to continue" x=250 y= height - 20);
```

```
//intro2
```



```
CALL loadAlien    //loads an alien image
```

```
CALL removeAlienBackground //removes the
                           background
```

```
CALL introAlien   //animate alien space ship to
                  middle
```

```
//playGame
```

```
DISPLAY the binaryShip object
```

```
//uses the show property of the binaryship object
```

```
MOVE the binaryShip object
```

```
//uses the move property of the binaryship object
```

```
CALL displayScore
```

```
CALL startAliens
```

```
CALL fireBullets
```

Level 3 pseudocode

```
//loadAlien – this function loads alienImage into an empty pixel array
```

```
LOAD alienImage pixels into a two-dimensional array
```

```
CREATE an empty two-dimensional pixel array w = 235 h = 214
```

```
LOAD alienImage pixels into empty transfer array called alienImageTrans
```

The problem here is that the image file used for the alien ship has a white background. We need an algorithm that removes the white background. Let's explore image pixels for a moment so we can develop an algorithm for this. Figure 1.4.7 is a black-and-white image and so only takes 1 bit per pixel. The illustration shows what happens when an image is loaded into a pixel array. Figure 1.4.7 demonstrates the RGBA storage of a colour image which is used for the alien space ship.

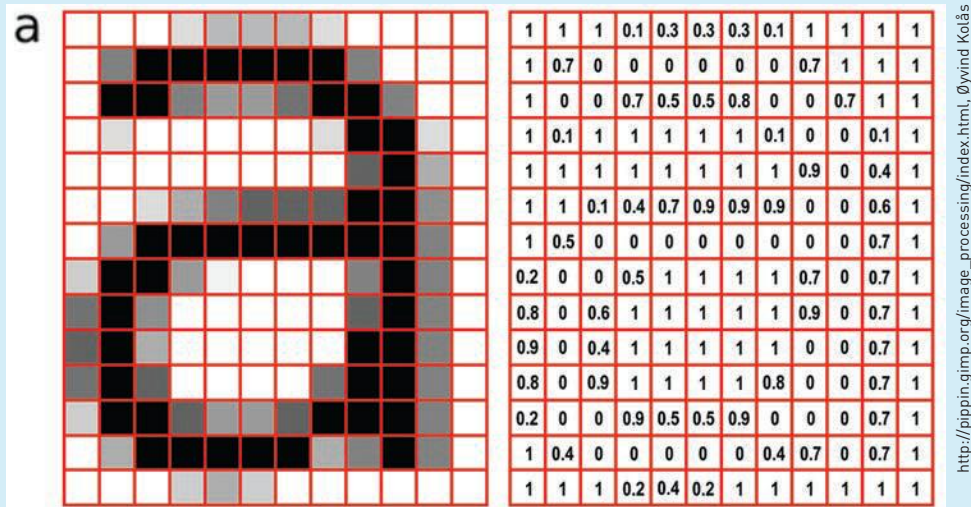


Figure 1.4.7 Scaled pixel image array

We know that a white pixel has the value $r = 255$, $g = 255$, $b = 255$. An algorithm could be developed to iterate through the pixel array for occurrences of white. One small dilemma shown in figure 1.4.8 is that every fourth pixel holds the transparent value. The algorithm must skip every fourth pixel in the array. Mathematics provides us with a way to do this using this formula:

$$i = (x + y * \text{alienImage.width}) * 4$$

Now we can develop an algorithm to remove white.

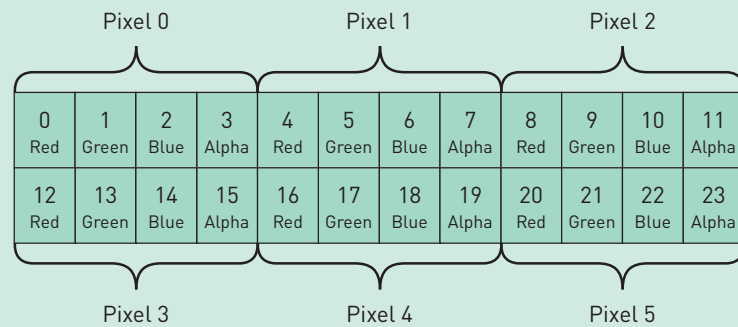


Figure 1.4.8 Colour pixel image array

```
//removeAlienBackground removes white background from alienimage and scales down to 50px across
FOR y = 0 to y < alienImage.height y++ DO //moves down the pixel array
  FOR x = 0 to x < alienImage.width x++ DO //moves across the pixel array
    i = (x + y * alienImage.width) * 4 //this skips the alpha pixel in the rgba order
    IF alienImage.pixels[i] < 255 AND alienImage.pixels[i + 1] < 255
      AND alienImage.pixels[i + 2]
    < 255 THEN //copies only rgb pixels that are not 255 into alienImageTrans array
      alienImageTrans.pixels[i] = alienImage.pixels[i]
      alienImageTrans.pixels[i + 1] = alienImage.pixels[i + 1]
      alienImageTrans.pixels[i + 2] = alienImage.pixels[i + 2]
      alienImageTrans.pixels[i + 3] = alienImage.pixels[i + 3]
    ENDIF
```



Explore the colour matrix in action. Try changing some of the colour settings to see what happens.


```

UPDATE pixels into alienImageTrans
RESCALE alienImageTrans to 50px wide

//introAlien animates the flying alien out to the middle of the canvas.
IF alienCount < width / 2 THEN           //stops moving when alien in the middle
    alienCount = alienCount + 2           //increments x position of alien
    alienImageTrans x = alienCount        //move alien out to middle
ELSE
    CALL displaySecretCode
ENDIF

//displayScore is called by function playGame and displays ammo usage
DISPLAY 'Binary bullets used:' + amo at x = 10 and y = 50
DISPLAY 'Score:' + score, x = 300, y = 50
DISPLAY 'Kills:' + kills, x = 500, y = 50

//startAliens is called by playGame and shows and moves aliens, detects if aliens hit the edge of the
//canvas and moves them down and in opposite direction
DECLARE edge = false                     //flag to indicate if aliens hit the edge
FOR i = 0 to i < asciiAliens.length i++ DO //loops to size of the alien array
    //display and move the array of aliens

    DISPLAY asciiAliens[i]
    IF kills < 8 THEN                     //this only lets aliens move if not all killed
        MOVE asciiAliens[i]
    ENDIF
    IF asciiAliens[i].x > width or asciiAliens[i].x < 0 THEN
        edge = true                       //detects when aliens hit edge of canvas
    ENDIF
    IF edge is true THEN
        FOR i = 0 to i < asciiAliens.length; i++ DO
            MOVE asciiAliens[i] down      //move aliens down
        ENDFOR
    ENDIF
    IF kills < 8 THEN
        score = score - 2                 //lose 2 points if aliens move down a level
    ENDIF
ENDFOR

//fireBullets moves ship and fires bullets and checks for a hit
FOR i = 0 to i < length of bullet array i++ DO
    SHOW binaryBullets[i]
    MOVE binaryBullets[i]
    FOR j = 0 to asciiAliens.length j++ DO
        //checks if hit and if correct key was pressed
        IF bullet hits Aliens AND correct key was fired THEN
            IF alien not hit THEN         //stops hitting already hit aliens
                score = score + 5         //5 points for a kill
                kills = kills + 1         //increment kills
            ENDIF
        ENDIF
    ENDFOR
ENDFOR

```

```

        Remove alien          //make radius of alien 0
        Remove array bullets  //flag to remove old bullets from the array
    ELSE
        //remove bullets from array that have hit
        FOR i = binaryBullets.length - 1 to i >= 0 i-- DO
            IF binaryBullets[i].toDelete THEN
                binaryBullets.splice(i, 1)
            ENDIF
        ENDFOR
    ENDIF
ENDFOR
ENDFOR

```

Level 4 pseudocode

//**displaySecretCode** – animates a beam of light from the alien and generates a random letter for the player to then convert to binary

```

    DISPLAY alienImageTrans at x = width of screen / 2 y= -20          //display the alien
    FILL colour white and semi transparent
    OUTLINE colour green
    IF beam < height - 20 THEN          //stops the animation when reaches bottom
        MAKE a triangle with height = beam
        beam = beam + 1
        DISPLAY a random character in middle of triangle
    ELSE
        DISPLAY a triangle at full height
        DISPLAY 'Greetings earthling' at x = width / 2 - 70, y = height - 70
        DISPLAY 'There are eight aliens' at x = width / 2 - 70, y= height - 50
        DISPLAY 'The secret symbol is' x = width / 2 - 70, y= height - 30
        OUTPUT secretSymbol at x = width / 2, y = height - 10
        DISPLAY 'press any key to continue' at x = 50 y = height - 20
    ENDIF

```

The final piece of the development puzzle is considering the event listeners that continually listen and will respond to events that happen in the game. The events in this game are keyboard events. In particular, you will be interested in steering the ship and firing your bullets. Here we consider what happens, or could happen, when an event is triggered. For example, if a bullet is fired, what can possibly happen? This is the focus of the event-driven algorithms.

Let's now unpack the keyboard event algorithms. We need to consider the following events: if the right arrow is pressed, the ship will move right; if the left arrow is pressed, the ship will go left; if number 0 is pressed, the binary bullet 0 is fired; and finally, if number 1 is pressed, the binary bullet 1 is fired.

```

keyPressed()          //listens for keyboard events
    //shoot bullets
    IF key = '0' THEN
        DECLARE new binaryBullet at x = binaryShip.x, y = height, value =0    //create new
        //binary bullet object
    
```

ACTIVITY 1.4.8

Imagine you were an event listener in a tic-tac-toe computer game. A human arrives to play you. List the events that might take place once the game starts.



```

        ADD the binaryBullet to the binaryBullets array           //adds a bullet to the array
        amo = amo + 1                                             //keep track of bullets used
        score = score - 1                                         //take one from score per bullet
    ENDIF
    IF key = '1' THEN
        DECLARE new binaryBullet at x = binaryShip.x, y = height, value =1
        ADD the binaryBullet to the binaryBullets array
        amo = amo + 1
        score = score - 1
    ENDIF
    //steer ship
    IF keyCode = RIGHT_ARROW THEN
        binaryShip.setDir(1)                                     //sends the direction to the ship
    ELSEIF keyCode = LEFT_ARROW THEN
        binaryShip.setDir(-1)
    ENDIF

```

Once all algorithm ideas have been exhausted and desk checking is complete, it is time to code. This game is played in a web browser, therefore an HTML file is required to link all components together. The following is the HTML in index.html:

ACTIVITY 1.4.9

Perform a desk check on the keyPressed algorithm. Refer to section 1.3 for the process.

```

<!DOCTYPE html>
<html>
<head>
    <title>Ascii Invaders</title>
    <!-- link all external sources -->
    <script src="https://cdnjs.cloudflare.com/ajax/libs/p5.js/0.5.8/p5.js" type="text/
        javascript"></script>
    <script src="js/binary_ship.js" type="text/javascript"></script>
    <!--library-->
    <script src="js/ascii_alien.js" type="text/javascript"></script>
    <!--library-->
    <script src="js/binary_bullet.js" type="text/javascript"></script>
    <!--library-->
    <script src="js/introduction.js" type="text/javascript"></script>
    <!--library-->
    <script src="js/game.js" type="text/javascript"></script>
</head>
<body>
</body>
</html>

```

You will see that in the HTML above, there is nothing in the body section. That is very different from the last case study as everything is being run within JavaScript files. We just need the HTML file since it is a web game. We are using the p5.js JavaScript library as part of the development; it simplifies generating HTML elements directly from JavaScript such as the canvas. Only the game JavaScript file runs the game. The others are storing all the functions relating to that object.

The development environment being used is Visual Studio Code. Refer back to section 1.1 for an explanation of the setup. The first step is always to set up the folder structure, which was explained in the last case study. In this case study, likely folders would be:

- 1 JavaScript
- 2 Images
- 3 Audio
- 4 Libraries.



To view the complete generated solution for ASCII Aliens go to NelsonNet.

TEST YOUR KNOWLEDGE

- 1 Describe some advantages and disadvantages of using external libraries and frameworks.
- 2 Identify three organisations in information technology that aim to support global standards, and identify in what context this takes place.
- 3 Explain the term 'full stack development' in the computer industry.
- 4 List five recommended steps to take when setting up a development environment for starting a new project.
- 5 What does 'version control' mean?
- 6 Explain the purpose of CSS.
- 7 Explain the purpose of the head section of a HTML web page.
- 8 Develop a list of instructions for someone new at web development, explaining how to link a Google Font to their web page.
- 9 Repeat question 8 for icons.
- 10 What approach is used to make a website responsive? Explain how this works.
- 11 Explain the CSS box model.
- 12 What details does a class diagram show?
- 13 What is the difference between a class and an object?
- 14 What type of errors are normally detected in a desk check?
- 15 Explain the difference between local and global variables, and how and when they are used.
- 16 What is a parameter?
- 17 When would you use an array?
- 18 Explain why a colour image stored in RGBA format takes 4 bits of information, as opposed to a black-and-white image, which only takes 1 bit.

APPLY YOUR KNOWLEDGE

- 1 Opaque green has the following RGBA colour values:

```
rgba(51, 170, 51, .1) /* 10% opaque green */
rgba(51, 170, 51, .4) /* 40% opaque green */
rgba(51, 170, 51, .7) /* 70% opaque green */
rgba(51, 170, 51, 1) /* full opaque green */
```

 Based on this knowledge, modify the existing code explored in this section for colour removal to remove all green from an image with opaqueness less than 40 per cent.
- 2 On 1 January for each of three years, a woman invests \$200 in a fixed-income bond that pays interest of 8 per cent per annum, the interest being added to her account at the end of each

year. The following algorithm gives the total value of her investment after the three-year period.

```
A = 0
FOR I = 1 TO 3
    A = A+200
    A = 1.08 × A
```

```
NEXT I
PRINT A
```

- a Trace the algorithm.
- b Explain how the algorithm could be amended to display the value of the investment at the end of each year, after interest for that year has been added.

- c Develop a modified algorithm that would find the value, after a period of N years, of an investment of $\$X$, invested at the start of each year, in a bond paying a constant rate of interest of R per cent per annum.

- 3 The algorithm below is used to generate a sequence of numbers.

```

LINE 10    INPUT A, B
LINE 20    PRINT A, B
LINE 30    WHILE C < 10 DO
LINE 40        LET C = A + B
LINE 50        PRINT C
LINE 60        LET A = B
LINE 70        LET B = C
LINE 80    END WHILE

```

- a Trace the algorithm when $A = 1$ and $B = 1$.
b Suppose that LINE 30 is changed to WHILE $C < 50$

Write down the extra values that C now takes.

- c A student mistypes LINE 70 as LET $C = B$. Find the values of A , B and C that the student would get using the amended algorithm.

- 4 The following algorithm is to be used on different numbers.
(The function INT gives the integer part of any number, for example $\text{INT}(2.5) = 2$)

```

LINE 10    INPUT N
LINE 20    LET K = 0
LINE 30    WHILE N > 0
LINE 40        LET A = N - (2 * INT (N / 2))
LINE 50        LET K = K + 1
LINE 60        PRINT A
LINE 70        LET N = INT (N / 2)
LINE 80    END WHILE

```

- a Show that, if the input value of N is 5, then the algorithm will print 1, 0, 1.
b Trace the algorithm in the case where the input value of N is 12.
c Find the final value of K in the case where the input value of N is 20.

- 5 Consider the following problem, known as the 'Tower of Hanoi'. Use the explore, develop, generate, evaluate process to develop an algorithm for this problem:

You have a stack of discs, from largest to smallest, that slide onto the first peg of a three-peg board. Your goal is to move the entire stack of discs from the first peg to the third peg. However, you can only move the topmost disc from any peg, and smaller discs must always be placed on larger discs. How many moves will it take?

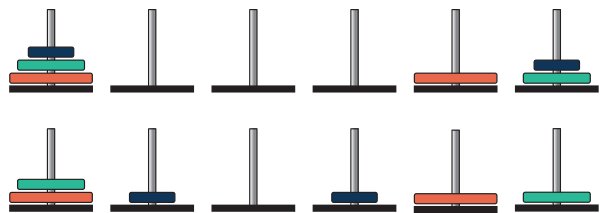


Figure 1.4.9 Tower of Hanoi

- 6 Use your Digital Solutions problem-solving skills to expand the ASCII Aliens game to a game for two players. This will require two scores to be displayed, one for player A and one for player B. Both players use the keyboard.
7 Build on question 6 to add another level to the game, which will involve two rows of aliens. This will require you to display two secret codes before the new level commences.
8 Develop your own colour-removal digital solution that accepts an image stored in a file and removes a colour based on user selection on the screen. You will need to develop your own user interface. You will also need to modify the algorithm provided in this section to generate your final code. Develop a useability checklist and evaluate the solution from a developer and user perspective. Collect user feedback.

Unit

2

APPLICATIONS AND
DATA SOLUTIONS

INTRODUCTION

In Unit 2, students will build upon their experiences in Unit 1 by exploring the incorporation of structured data into their digital solutions. Using the various phases of the problem-solving process, students will explore and be prepared to generate data-driven solutions using self-developed and partially normalised datasets. Students will explore the normalisation process to further refine databases and meet solution useability and reliability requirements. Key to this unit is investigating and understanding the viability, features and structure of a relational database including the use of primary keys, foreign keys, field data types and data constraints to maintain data integrity and reduce issues such as data redundancy.

Solutions will be explored and evaluated against criteria relating to suitability, reliability, impacts and data principles. Students will investigate data definition and manipulation processes by learning Structured Query Language (SQL) and its application within the chosen technology contexts. Students will be provided with opportunities to incorporate SQL within text-based code scenarios to determine the interconnectivity between application layers and data systems. This will provide opportunities for further learning experiences in developing data-driven digital solutions in chosen technology contexts.

Students will evaluate the issues and impacts against key concepts relating to the use of data and information in data-driven digital solutions. This includes exploring and evaluating issues and impacts relating to security, integrity, privacy, storage, growth and ethical considerations.

KEY IDEAS

- Data flow relationships
- Programming elements
- Digital systems
- User interface
- Visual communication principles
- Visual communication elements
- Programming information
- Ideas
- Data stores
- User experiences
- Problems
- Information
- Technology context
- Requirements
- Prescribed criteria
- Self-determined criteria
- Prototype digital solution
- Impacts



2.1



EXPLORING DATA-DRIVEN APPLICATIONS AND SOLUTIONS

In this section, concepts around data-driven applications and solutions will be explored. The characteristics of data will be investigated and how data gathered together from a range of sources in a useable context can provide meaning and value to individuals, society and the economy. Aspects relating to the presentation of data, and how elements and principles of visual communication can be used to enhance user experiences of data-driven solutions, will be investigated. Students will explore how information systems can be viewed from different perspectives and how developers must work to successfully generate databases for uses in integrated data-driven solutions. Students will also explore the features of relational tabulated data and will include the identification of features such as fields, records, keys and constraints. Issues relating to the management, validity, privacy and security of data will also be canvassed to provide opportunities to apply this knowledge in future sections.

Data, information, knowledge, wisdom

Data is all around us. Every modern-day process that we take for granted involves the use of data, and this has allowed our society to be extensively data-driven. Today, the sheer amount of data stored across the world is beyond anything previous generations could have imagined. Every element of information can be captured, represented, manipulated and sorted for a variety of uses.

Data refers to a collection of facts that we might store, manipulate or use – for example, the length, width and height of a product could be collected as 200 mm, 100 mm and 50 mm. Our names can be thought of as data, as can other elements such as our date of birth, hair colour, height, weight and contact details. When linked together, data can be structured to form useful information that can be accessed for any required purpose.

To explore this phenomenon, let's investigate an imaginary takeaway restaurant gourmet burger item to understand concepts of data and information.



Sam's Breakfast Bar

The menu item shown in figure 2.1.1 represents a large set of individual data elements. These are sometimes referred to as data facts. The price (\$6.25), an ingredient (sugar) and grams of protein per serve (21.3) are all examples of this. Note that they are clear and definitive facts that do not have an ambiguous meaning. This is because of the descriptive nature that is associated with the data facts. For example, the description 'price' gives associated meaning to the data '\$6.25'. The descriptive term 'ingredient' gives associated meaning to 'sugar' and so on. This becomes important as we bring pieces of data together to give them meaning. When we store data, these descriptions are used as identifiers, as distinct from values. To understand this concept in a different scenario, in a school library system the data '*The Adventures of Huckleberry Finn*' would be identified as the 'title' of a book, and the data '384.234' would be identified as its 'shelf location'.

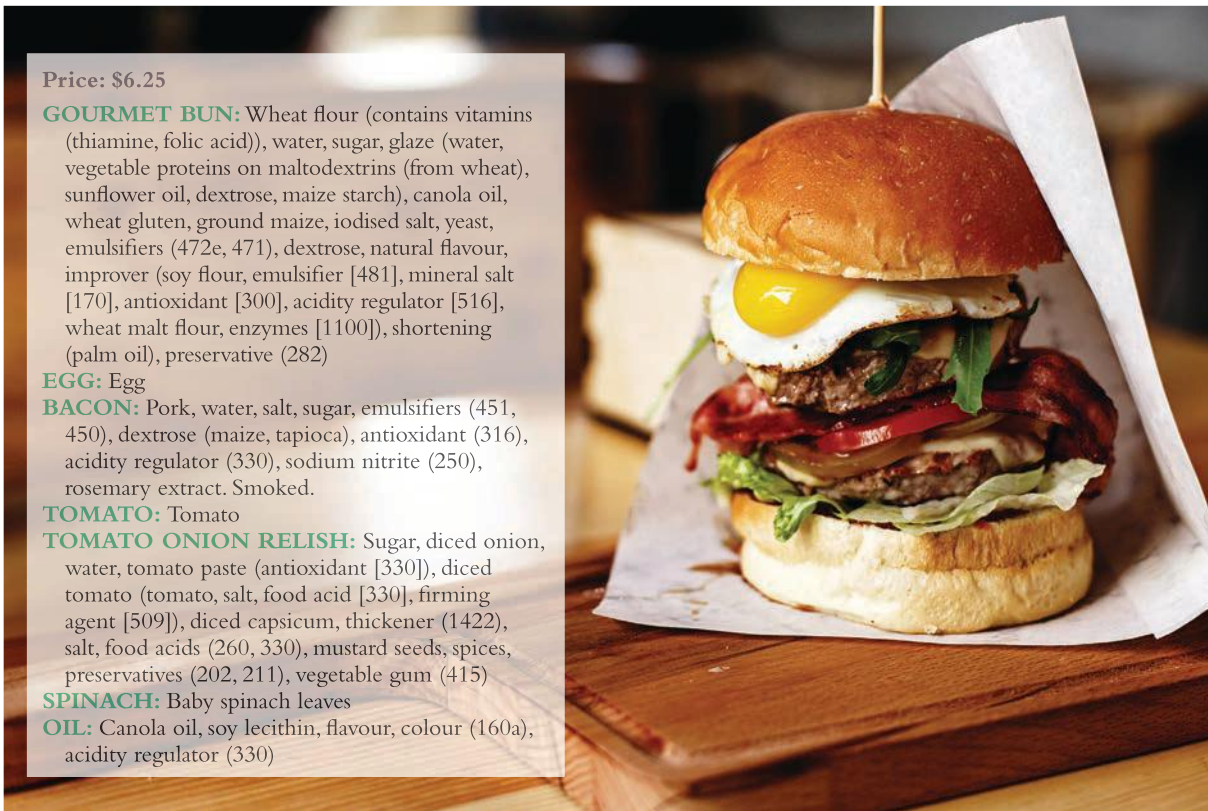


Figure 2.1.1 Information about a single menu item consists of many individual data elements.

Table 2.1.1 Data list of ingredients of the gourmet burger – a tabulated data example of nutrition measurements

NUTRITION	AVERAGE QUANTITY PER SERVE	AVERAGE QUANTITY PER 100 GRAMS
Energy (kJ)	1730.0	814.0
Protein (g)	21.3	10.0
Fat, total (g)	12.5	5.9
Fat, saturated (g)	3.9	1.9
Carbohydrate (g)	51.5	24.2
Sugars (g)	14.9	7.0
Sodium (mg)	986.0	464.0

GROUP ACTIVITY 2.1.1

- 1 As a group, discuss the value of this data as information. In what situations could it be used?
- 2 Brainstorm potential digital solutions in which this data might be used to help people.
- 3 Individually, select one of the brainstorming ideas and prepare an 'elevator pitch' – a one-minute product pitch.
- 4 Present your pitch to your group.
- 5 As a group, select one idea and explore how it may be experienced by the end user. Sketches, annotated diagrams and features listed as bullet points are suitable approaches to this task.

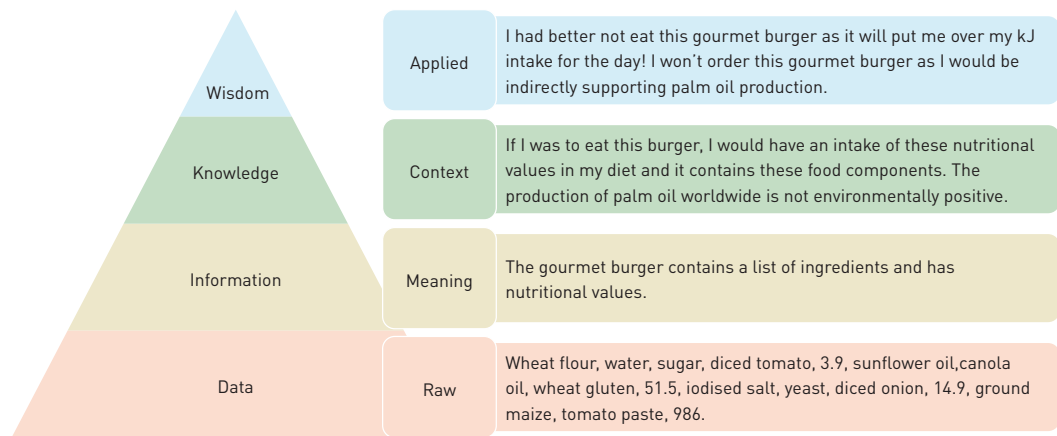


Figure 2.1.2 How data becomes meaningful

Figure 2.1.2 illustrates how data becomes meaningful to the user when it is assembled, structured and related together. The pieces of data become recognisable as a whole and take on greater meaning. When related data is gathered together in this way, it can be thought of as information.

Information is a grouping of data that can collectively represent meaning and become a useful source. In the case of the burger menu item, the different ingredients and nutritional measurements come together to form information about the burger. They can be recognised as a whole. They have gained further meaning by being placed together, rather than just being individual data elements on their own. The combined data collected together as information in a data store to represent a topic is sometimes referred to as its **universe of discourse**.

'Universe of discourse' is a phrase that refers to a limited set of real-world data that has been chosen to be collected and stored to meet a specific purpose. In a world full of data, only data that directly relates to the topic or theme – the universe of discourse – must be used in the solution. For example, data about students – their name, address, subjects, results, medical details and who their parents are – can be gathered together in a school management database. Data about what toys each student has at home or the rules of their favourite sport would be external to this topic.



As figure 2.1.2 shows, it is then possible to review the nutritional information of the gourmet burger to understand the make-up of the product and to interpret levels of energy, fat, sugar and sodium in a serve. This can provide a context to the meaning of the information. At this stage, the contextualised use of information can be considered useful knowledge.

Using knowledge to make decisions – for example, what food is acceptable for an individual based on nutritional knowledge and knowledge of the user's tastes or recognition of issues – is known as wisdom. This is the next evolutionary step in the use of data illustrated in figure 2.1.2: the development of wisdom from knowledge. This hierarchical approach to understanding data through to wisdom can be applied to the actions undertaken in various societal functions every day. Wisdom is the ultimate goal in the application of data and, with the explosion in the amount of data available, data-driven solutions can assist in supporting this.

ACTIVITY 2.1.1

- 1 What is meant by the term 'information'?
- 2 How do accurate and inaccurate information influence wisdom? Explain using the data-information-knowledge-wisdom model.
- 3 In pairs, think of three possible professions where wisdom is considered highly valueable. How might an existing or new data-driven system support these professions?
- 4 Some commentators proclaim that society has now entered the information age. What does this term mean and how does it relate to the data-information-knowledge-wisdom model?

Sources and uses of data

For many years the standard way to enter data has been to fill in a form, for example when applying for a driver's licence or opening a bank account. Such forms gather information about individuals, and allow organisations to link this data to a variety of external sources. These sources contain rich data such as tax file numbers, birth certificate identification codes and passport numbers.

Smart technology, with built-in sensors that read data and store it in memory, has increased dramatically over the past 50 years, allowing access to huge datasets that can be managed and retrieved in a variety of scenarios.

Government agencies around the world have made efforts to make data available to the public for use – for example, to find out if a car parking space is available in the city. The concept of 'open data' is that data should be freely available for use in systems and services for the betterment of society. In the case of the Australian and Queensland governments, open datasets cover tabular, spatial and relational data relating to a range of areas.



Figure 2.1.3 Australian taxation declaration form used to source information from individuals relating to paid employment

iStock, Unreleased/
Getty Images Plus/Owengat

GROUP ACTIVITY 2.1.2

In groups of two or three, explore an open dataset from an Australian source. For example, use the Australian Government open data website to identify:

- the purpose of the data
- features of the data
- where it has been collected from.

Brainstorm innovative ways of using the data. Sketch two different ways in which the data could be presented to a user. Identify which presentation method you would choose, and why. Share your sketches with another group. Would they choose the same presentation method? Discuss with them why or why not.



Explore



2.1.1
Australian
Government:
public data
2.1.2
Parking
infringement
data

ACTIVITY 2.1.2

The Bureau of Meteorology (BOM) relies on very powerful systems that use data to develop forecast predictions to try and keep citizens and assets safe. Explore the article listed and, in teams, complete the following tasks.

- 1 How does the Bureau of Meteorology gather base data for its weather systems?
- 2 Find some examples of datasets (historical data files) listed on the BOM website.
- 3 How is information presented to end users? Describe some examples.
- 4 How has longitudinal data (data collected over time) assisted in forming knowledge in this data scenario?
- 5 Is wisdom being drawn from data, information and knowledge? If so, identify a situation that might explain this.



Explore



2.1.3 New
computer to
supercharge
weather
forecast
2.1.4
Bureau of
Meteorology

GROUP ACTIVITY 2.1.3

In teams of two, develop an enrolment form for Year 11 and 12 students. In your teams, think about the type of information that the school will need from the student and their family. Suggestion: Identify datasets in groups (personal information, subject choice etc.).

Swap your solution with a peer group and evaluate the effectiveness of the data collection tool.



Develop



Evaluate

CASE STUDY

EXPLORING WEB-BASED CONTEXT: EXAMPLE SCENARIO

Garden Express, previously known as Bulb Express, has become one of Australia's largest online and mail-order gardening services. This case study is a simulated review of the transition of the company from a mail-order service to fully online sales and ordering service.

Background

Bulb Express has recently expanded its range and transitioned the company to a new name – Garden Express.

The company is situated in regional Victoria's Yarra Valley adjacent to the Monbulk–Silvan horticultural area. The area has a long history as a source of both ornamental and produce crops and the company has tapped into local knowledge to provide garden-based products and advice to its customers. A mail-order service was established to promote and gain a greater client base across Australia. Following the success of this aspect of the business, the team is looking to expand its brand and product range into a purpose-built online web store. The aim is for Garden Express to become Australia's leading online and mail-order gardening service and to offer the widest range of quality garden products, including living plants, seeds, bulbs, garden tools and accessories.

Purpose of this example

Throughout this unit, the Garden Express web scenario will be used as an example to model the process of exploring, developing, generating and evaluating a data-driven solution. It should be noted that this example is a possible simulated response to a real-world product for the purposes of learning, and may not reflect the true underlying components of the real Garden Express web store.

Defining the problem

The team at Garden Express has consulted with an online store developer. The developer embarked on a brainstorming exercise to understand the problem.

Determining the requirements for a digital solution involves identifying the **functional requirements** (what it needs to do), the **non-functional requirements** (how it needs to do it)

and the **constraints** of the user and developer environments. In determining these elements, a mind map has been constructed to symbolise the requirements of the data-driven problem.

Identifying these aspects of the problem has formed a list of requirements that can be used throughout the project as criteria for evaluation. Through the exploration of the problem and the completion of the mind-mapping exercise, the following requirements have been determined:

Requirements (functional)

- Customers must be able to setup online accounts and manage details, including names, addresses, email, passwords, delivery notices.
- Customers must be able to use their email as their username.
- Customers must be able to reset forgotten passwords independently.
- Display each product by group, category, sub-category.
- List each product – name, product code, price, image, details, advice, further information.



Figure 2.1.4 Garden Express home page

The process of defining the problem is further detailed in Unit 3.



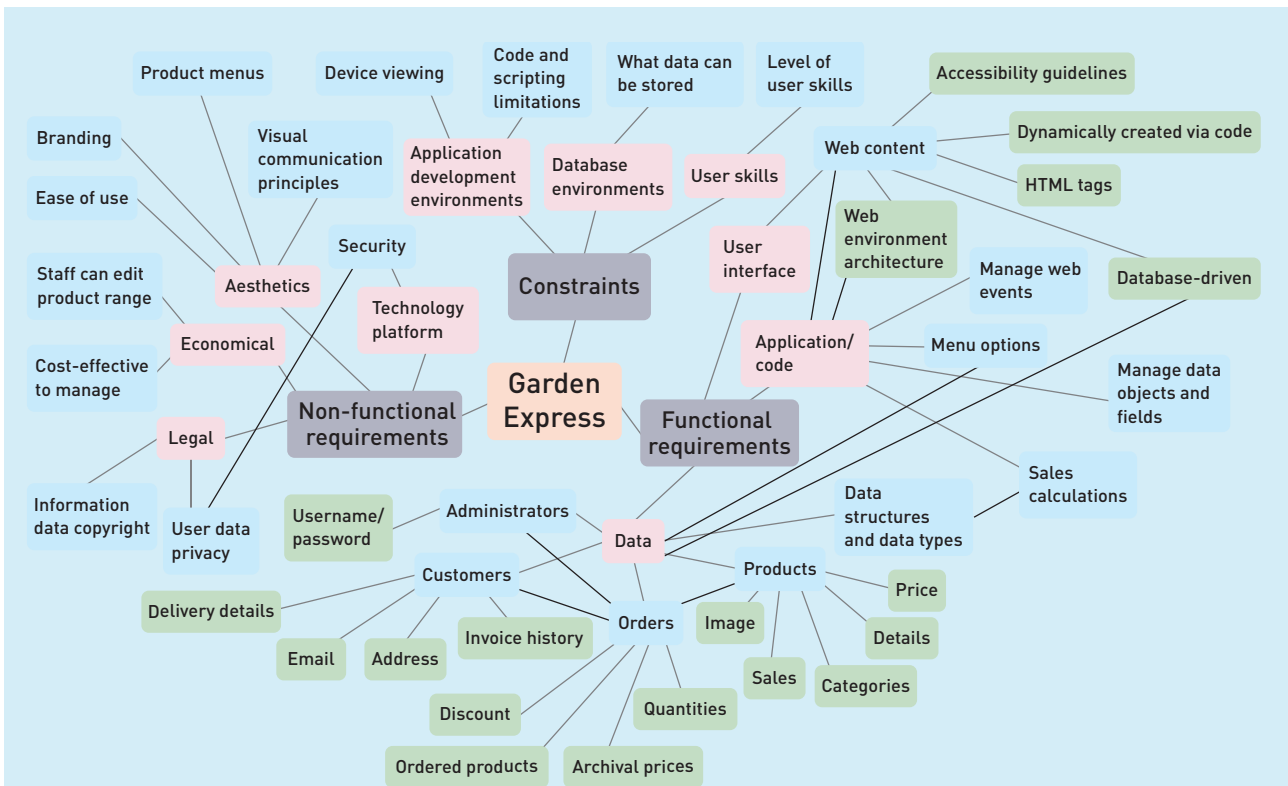


Figure 2.1.5 Garden Express problem mind map

- Allow customers to create orders, including adding delivery details.
- All customers to add, remove and adjust items in their order before they are committed.
- Allow customers to pay for their orders using financial credit card services.
- Allow manual processing of payments by employees.
- Record each order invoice to archive.
- All items must be able to be added as a sale item.
- Category groups, categories and sub-categories must be used in the navigation of the site to find products.
- Order totals and prices must remain historically accurate.
- Multiple items and quantities can be ordered in a single order.
- Employees need the ability to add items and to disable old items.
- Discounts on the overall price should be catered for. Employees may be able to apply this manually for the interim, but further functionality might be added for automatic promotional events in the future.
- Orders can be processed through a simple administration section.

Requirements (non-functional)

- It must be easy for users to seek products via navigation.
- Reflects branding of Garden Express Australia.
- Provides marketing opportunities.
- Images are large and support accessibility.
- The process for a user to sign up, log in and order is logical, supporting learnability.
- It is not too cluttered and does not look too similar to competitors.
- Uses a range of visual communication elements and principles to create a great user experience.

Working with data

Working with data provides opportunities to develop powerful digital solutions. Most people are aware that behind a web application, an ATM or a banking app is some form of logical data store. To explore and develop data-driven solutions, it is important that we understand the different ways to view and work with data. To do this, we will delve ‘under the hood’ using a simplified three-layer approach to data.

A three-layer approach to data

When exploring a data-driven system, the data within it can be thought of across three different viewing layers.

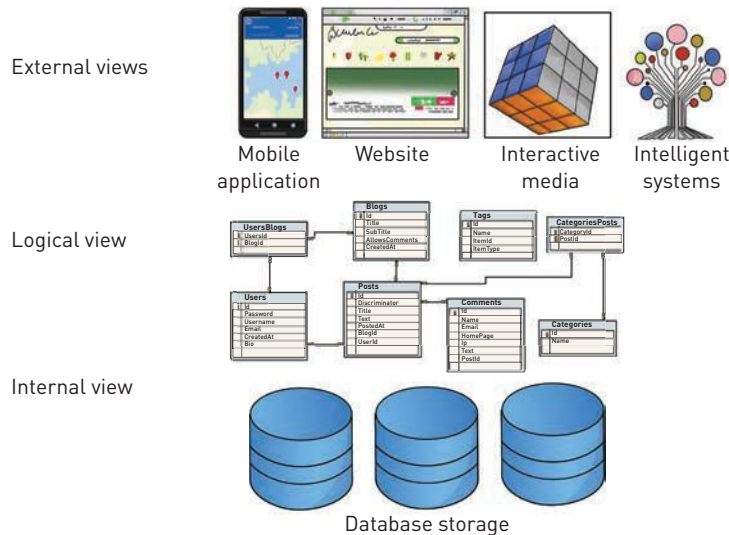


Figure 2.1.6 The three-layer approach to understanding data views in a data-driven system is applicable to all four technology contexts in Digital Solutions.

Internal layer view of data

The internal layer view of data relates to the underlying operational and physical storage and access structures used. This includes files, file systems, management systems and hardware operating systems that work in conjunction to serve and store data.

While it is important to understand the characteristics and requirements of data from an internal view perspective, most modern computing technologies automate these processes to allow designers and developers to focus on the logical and external viewing of data.

Logical layer view of data

The logical layer generally relates to how data is stored in ways that are well structured and maintainable. Looking at data characteristics in the logical layer can provide data-driven solution designers and developers with an understanding of how data needs to be collected to represent meaning, how it needs to be structured, and how data and information must interrelate with each other.

While there are a number of ways to represent logical views of data, their purpose is to communicate various aspects of how the universe of discourse is mapped into a well-managed, efficient set of structures, and what rules have been applied to ensure that the data remains a valid, reliable source of information over the life of the solution.

Data reliability refers to the ability to continually trust a data source and that the information accurately represents the facts, descriptions or actions it is representing. **Data validity** refers to ensuring data and information meet defined formats, rules and criteria to maintain integrity and reliability. **Data integrity** refers to the requirement that quality of data stored collectively is maintained to continue to have meaning and value.



Logical views of data are usually symbolic or structured representations that show links between data and give an efficient conceptual understanding of a data store such as a database. Developing a detailed logical view of data (such as tables) is usually part of a data storage development process and is a critical element in producing a viable data solution that maintains its information integrity over time.

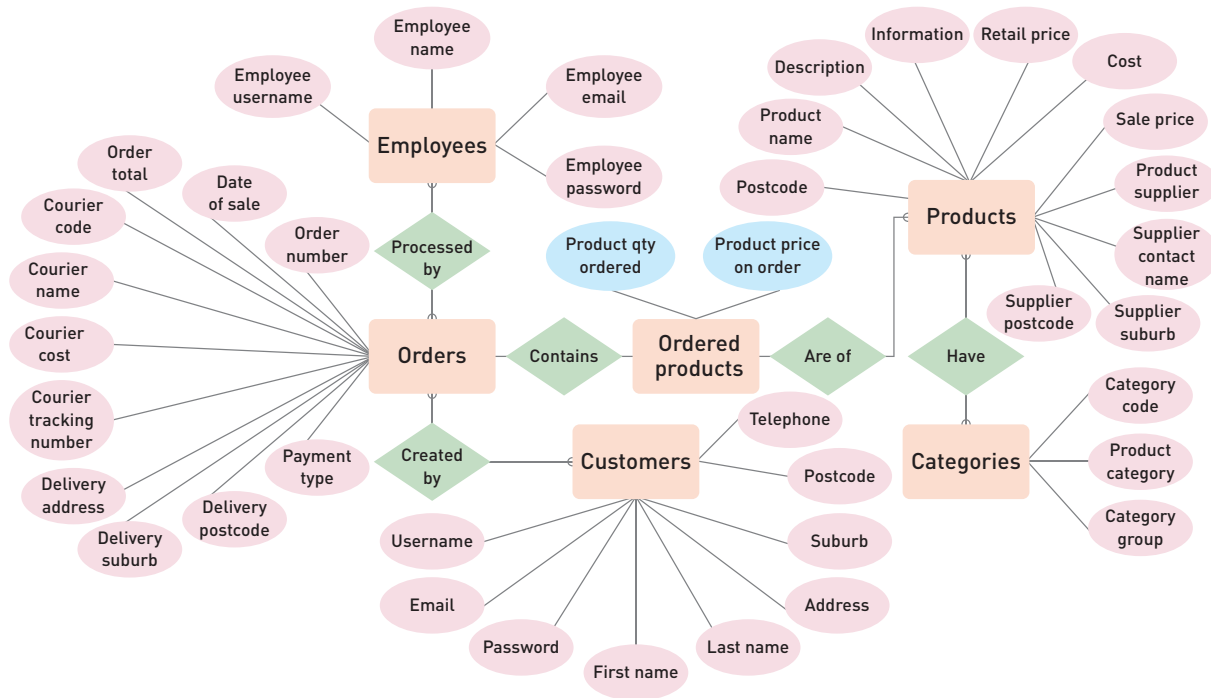


Figure 2.1.7 An entity-relationship diagram is a rules-based symbolic representation of how data is logically related within a database.

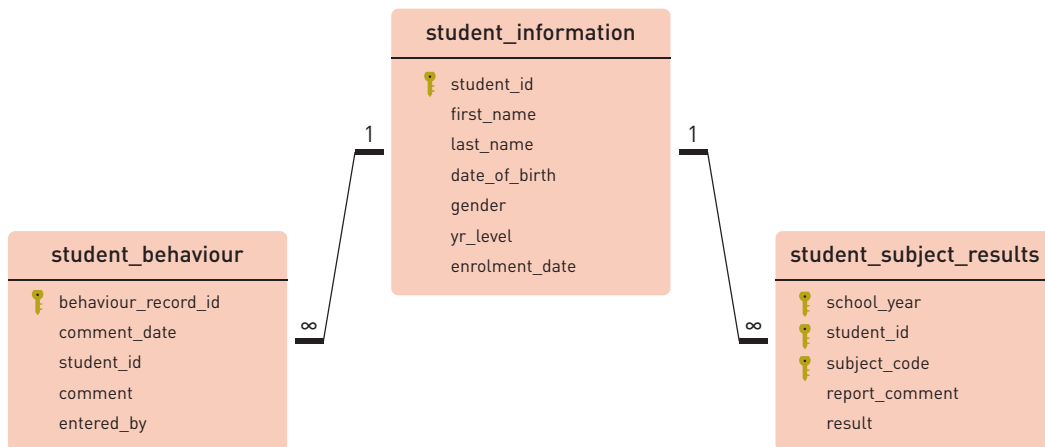


Figure 2.1.8 A relational database schematic details how data structures have been generated to store collections of interrelated data as information.

In later sections, entity-relationship diagrams will be explored, and relational schemas and database table schematics will be investigated and developed, to assist in generating databases as part of an overall data-driven system.

External layer view of data

The external data layer view of a system is what an end user might see or be witness to. This is when data is constructed and presented in such a way that users can interpret it and use it in suitable ways. As data-driven developers, it is important to think about how data can be presented to users and how they can effectively interact with it. In this view, designers and developers determine and specify what data will be accessible to users, how they can interact with it and what rights they have to access it or change it.

In a web, mobile or interactive media context, a constructed 'front end' of an application might be the way this is achieved. Data might be used in the form of list, grid or tabulated data, editable forms, graphical summaries or even as content of a screen that can change based on user interactions. It may also be a generated document that users can print or save. An example of this might be a bank statement.

In an intelligent system, data might be represented differently. As an output, it might take the form of visual elements such as light-emitting diodes, screens, sounds or movement. As an input, sensors may be collecting data that is processed and then stored.

courtesy Qantas

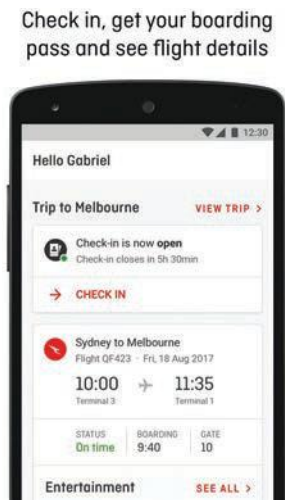


Figure 2.1.9 Example of an external view to a system. Accessible information has been presented using elements and principles of visual communication to enhance the user experience and interpretation of data.

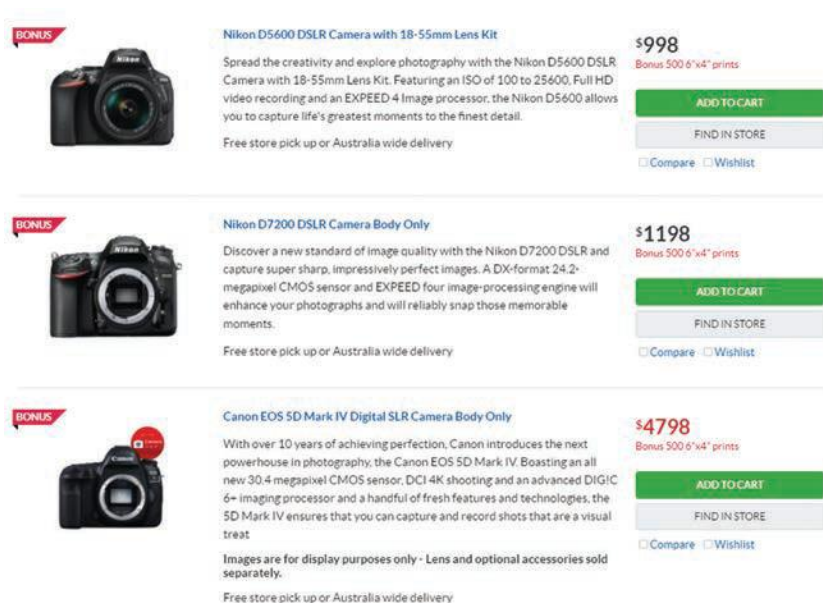


Figure 2.1.10 Data presented in a list form in a web-based store. Users select items that then link to different pages where more detailed product information is presented. This information is sourced from an underlying data store such as a database.

SparkFun

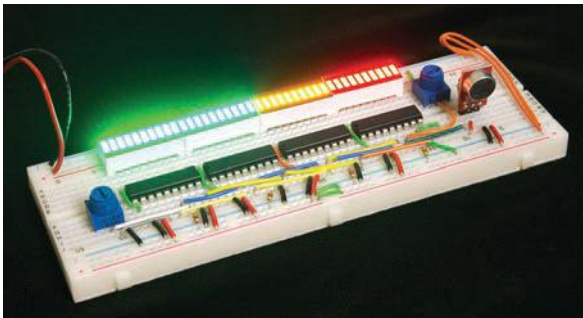


Figure 2.1.11 Arduino microcontroller using an LED matrix to output a graphical representation of data

ACTIVITY 2.1.3

Determine whether the following examples best reflect an internal, external or logical layer view of data:

- How effectively data is stored and accessed
- Representing the data in a way that communicates the intent of the universe of discourse of the data system
- Delivering data in a useful and authorised way for individual users.

Presenting data externally

The ways in which data is presented can support how a user interprets it as information and assist in making it meaningful.

As shown in figure 2.1.9, through the use of visual communication techniques, data can be organised within a user experience to greater support its use and understanding. By incorporating elements and principles of visual communication into the external layer view of data, data-driven designers and developers can greatly enhance user experiences and add additional value to the underlying data source.

Let's explore how data stored logically, such as in tabular form, can be manipulated and re-presented to enhance a user experience.

Exploring visual communication of data

When developing and generating a data-driven system, it is important to recognise and explore the elements and principles of visual communication. Employing these concepts within data-driven solutions can enhance a user's experience and also potentially support innovative ways to interface with data.

Figure 2.1.12 lists the principles and elements of visual communication that should be considered when thinking about developing, generating and evaluating a user experience for a data-driven solution. With any visual development, a heuristic approach is suggested as not all principles and elements will be viable or be applicable to every potential solution.

Elements of visual communication

Space

Space can be used to support meaning and to zone groups of data. It can be used consistently to develop predictability throughout a user experience. It should not be considered to be just negative or blank. An interface without the use of structured space could be considered by a user to be too busy, confusing and not learnable. The effective use of space can produce an aesthetically clean appearance, provide validity to interpreting data and is crucial to communicating with the user.

Artistic approaches to space can also be incorporated to engage users. This includes using space to provide a feeling and illusion of actual depth to give a greater three-dimensional aspect to experiences. Other concepts such as overlap and shadings can link visual elements within an interface to provide a seamless appearance to the layers of detail.

Colour and tone

The use of colour plays a significant role in developing data-driven applications and the presentation of associated data. Aspects of colour that should be considered include:

- a consistent approach to colour choice throughout the application. This assists in branding and the predictability of the product for users
- the compatibility of colours for ease of use. Some designers engage the colour wheel to identify compatible colour choices that support contrasting information as well as making the solution more accessible.

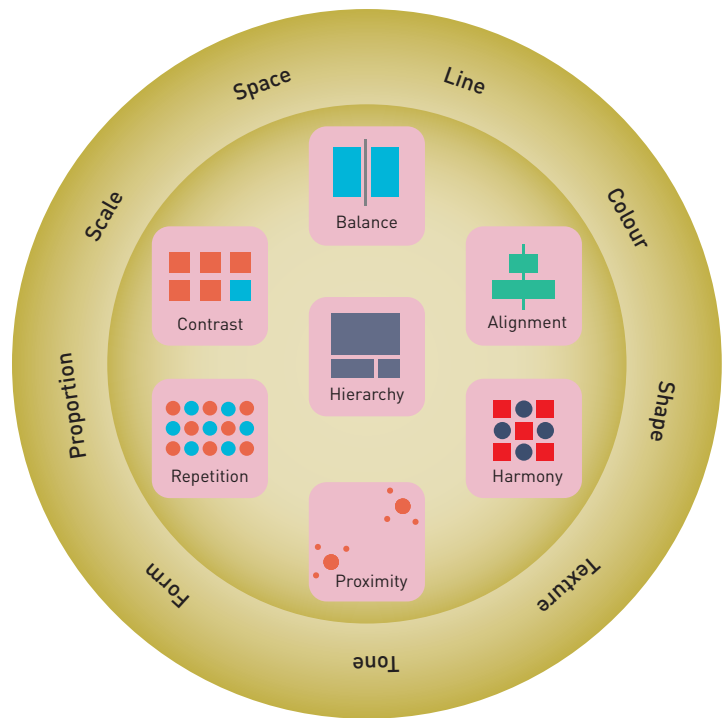


Figure 2.1.12 The principles and elements of visual communication

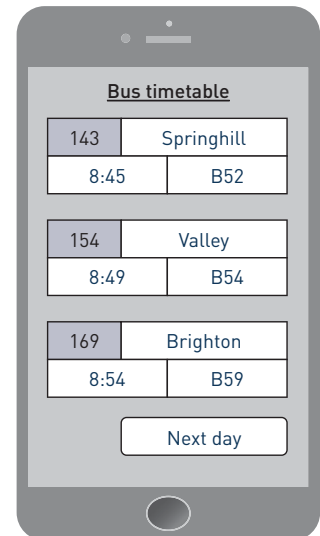


Figure 2.1.13 Space element used to distinguish instances of data – between buses

Show entries Search:

Username	Date registered	Role	Status	Actions
Bananaman	2011/06/01	Admin	Inactive	View Edit Delete
Batman	2011/02/01	Admin	Inactive	View Edit Delete
Bob the Builder	2011/02/01	Admin	Inactive	View Edit Delete
Catwoman	2010/01/21	Staff	Active	View Edit Delete
Fred Flintstone	2011/01/01	Member	Active	View Edit Delete
Garfield	2011/08/23	Staff	Banned	View Edit Delete
Hulk	2011/02/01	Staff	Banned	View Edit Delete
MacAndCheese	2011/03/01	Member	Pending	View Edit Delete
Paul	2011/03/01	Member	Pending	View Edit Delete
Ratamatat	2010/01/21	Staff	Active	View Edit Delete

Showing 1 to 10 of 16 entries First Previous 1 2 Next Last

Figure 2.1.14 Use of colour and tone to contrast data records, distinguish actions and emphasise data messages

Line, shape, texture, form

The use of the line element in interfaces is well defined and is naturally applied in the principles of alignment, proximity and hierarchy. A line can be thought of as a starting place, a marker or trigger to change. A simple horizontal line with textural and tone features within an interface such as a web page can provide separation or encapsulation. It is a fundamental element of design in every context and usually from an artistic perspective the starting point.

Shape can use lines (straight or curved) to develop two-dimensional zoning and the implied boundary of an object. Shapes are used regularly to convey structure around data. The simplest form is the table, providing a visual interpretation of the ordering and relationships of data to form records and combined meaning (information).

Shapes are also used in conjunction with visual interactive elements to centre a user's attention. A classic example of this is circular iconography in mobile tools, such as video recording on IOS, or menu components in an interactive media interface, or using shaped features for sensors to read or interact with.

Texture is a truly artistic element of visual design and refers to the tactile or inferred visual features of an object. Texture can be used to assist in visual or physical improvements in accessibility, or in conjunction with other elements such as form and shapes to give emphasis or distinction.

Utilising the element of form within a user experience can give the feeling of depth and a multi-dimensional perspective. This could assist in highlighting components through the use of shadows on the surfaces or faces of objects. Form is usually applied in conjunction with other elements such as tone, texture and colour. In relation to developing rich data-driven experiences, it may be possible to employ form to assist in setting the significance of data in a data-driven menu (for example, it layers down and appears deeper giving the impression of drilling down into specifics of data).

In figure 2.1.13, shape is used to centre a user's attention in relation to key data, but also to highlight navigation between day data schedules – the 'Next day' button.

Proportion and scale

Artistic design rules for proportion and scaling are also worth considering when implementing a solution.

Concepts such as the 'golden ratio' are used to demonstrate proportion in design work such as interfaces. This is a commonly occurring

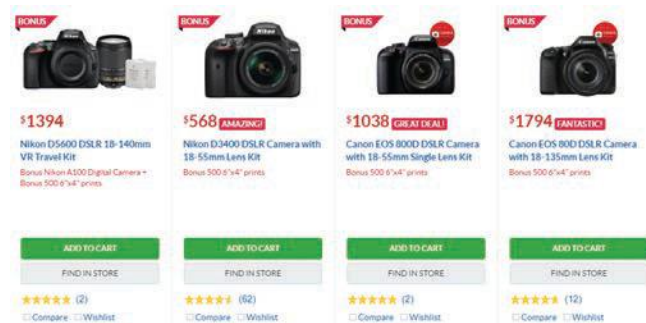


Figure 2.1.15 Use of line and shape to apply visual communication principles

mathematical ratio that has been discovered to create aesthetically pleasing, natural-looking presentations. For mathematicians, the golden ratio is nothing new and is closely related to the Fibonacci sequence.

To learn more about the golden ratio and how it can relate to data-driven visual communication techniques, please follow the weblink.

The use of concepts such as the golden ratio is evident in existing data-driven solutions such as Twitter and can be used in a visual structure to sub-categorise data and emphasise its importance to the user.

Fitts's law is a visual concept relating specifically to scale. To learn more about how Fitts's law could be applied to digital solutions, please follow the weblink.

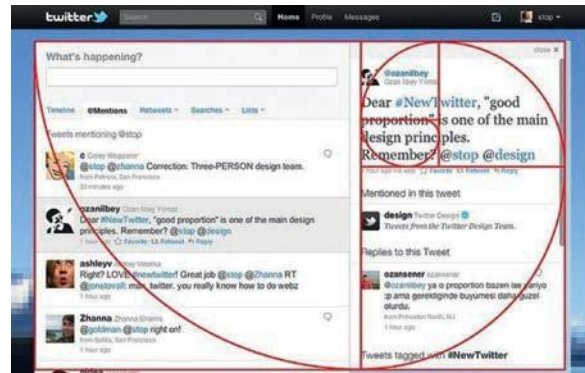


Figure 2.1.16 Proportion and scale used to delineate data-rich user experience – a known aesthetically pleasing approach



2.1.5 The golden ratio



2.1.6 Fitt's law

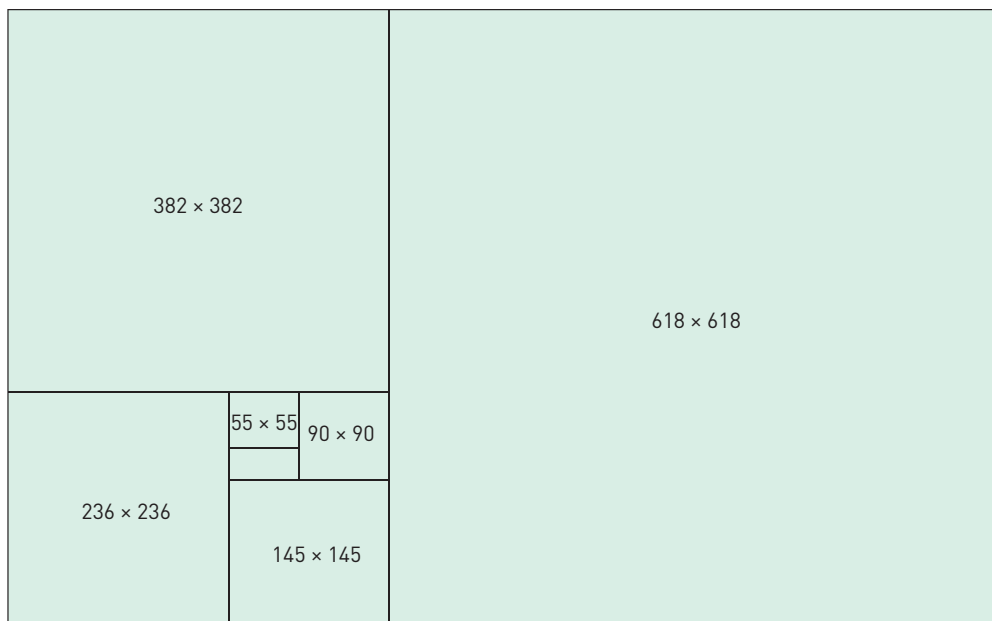


Figure 2.1.17 Example of the break-down of a web page into proportional zones based on the golden ratio

Principles of visual communication

Alignment

Alignment is a principle of design that is conducive to how users infer information from a layout. Alignment of images, text and objects provides structure, eliminates potential haphazardness and allows the user to effectively scan information. Alignment is one of the most fundamental and important principles of design and should be considered in almost every scenario. Traditional newspaper column structures are classic examples of how the design principle of alignment can be successfully deployed.

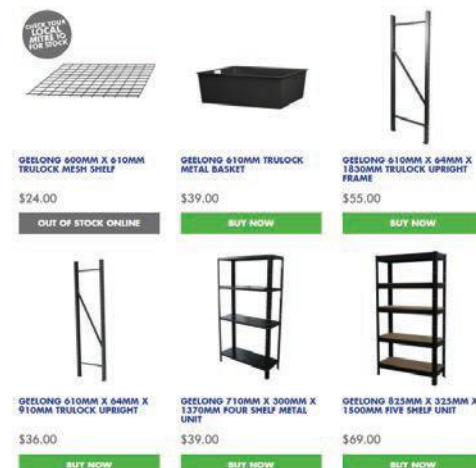


Figure 2.1.18 The principle of alignment in use

Mitre 10

Title

First Name *

Last Name *

Date of Birth *

Gender * ☐ Female ☐ Male ☐ Not specified

Email *

Confirm Email *

We will send communications such as statements, special offers, and program news to your email address, subject to our Privacy Policy.

Mobile

Figure 2.1.19 Alignment is commonly applied to data input screens

As shown in figure 2.1.18, the principle of alignment features heavily in data-driven websites. In this example, products are aligned using elements of shape and line. Elements of space, scale and proportion are also used to give further alignment that is aesthetically pleasing and that enhances grouping of data as product information.

Alignment is also commonly applied to data input screens to assist users by suggesting a logical sequence to entering information. As shown in figure 2.1.19, elements of shape, form, texture and tone are also incorporated to enhance the user experience.

Repetition

The principle of repetition provides predictability through the reusability of elements, and is commonly used in data-driven solutions. The repeated elements could be page constructs, sections or product layouts. This allows users to learn the user environment and predict where they will be able to source information they require. In a web context, product information and detail will appear in similar ways, and certain types of information will be placed in similar positions.

Datasets are often displayed in rows in a repetitive fashion. This is a principle of visual communication that is essential to tabulated data and provides uniformity, but it can become difficult to distinguish and follow if overused. As shown in figures 2.1.14 and 2.1.20, the visual communication principle of repetition is applied effectively by incorporating elements such as colour in alternating rows of the user list and the generated invoice.

Contrast

Contrast is what you see when you compare things that are different. Humans are wired to notice differences, and the clever use of contrast can support powerful interactions. Contrast can be achieved in simple ways by utilising elements such as colour and space, and in conjunction with other principles such as proportion and scale. Some designers argue that contrast is one of the most important principles of design and uses every single design element to achieve success. A simple menu system showing contrast between sections or sub-sections is a simple yet effective demonstration of this principle. In data terms, contrast could mean comparing components, or contrasting high-value data segments such as product placement, or distinguishing between products in a web store. In tabulated data, contrast can be a useful technique to indicate to a user which row is currently selected.

Hierarchy

Having clear, consistent and predictable levels of hierarchy in the presentation of the data-driven solution will greatly support the user's experience. This principle can be used to highlight the importance of particular content and features, encouraging users to respond to important elements. When designing a web page, it can be helpful to keep in mind the natural tendency of users to assign greater importance to content at the top left of the screen, and less importance as they move from the top down and from left to right.

In figure 2.1.21, the numbers provide a basis to identify the hierarchical features of the site. They show the levels of visual significance and reveal what the owners of the site wish to highlight to users.

Maxima ICT Services LTD

Off Kitengela Rd
P.O. Box 511
Nairobi NBI 00517
Kenya

+254 20 321 3220

+254 20 321 3230

<http://www.maxima.co.ke>

info@maxima.co.ke

TIN: KEP051534006T

Invoiced To:
Customer 001

123 Nairobi
Citizen Avenue
Nairobi NBI 00517
Kenya

1234567890

123456789

<http://www.customer001.com>

custo1@customer001.com



@Maxima_ICT

MaximaICT

+MaximaCoKeICT

Reference:
CX034

Invoice Date:
10-10-2016

Due Date:
9-11-2016

Fiscal Position:
Normal

Source Document:
S0004

Item	Description	Quantity	Unit price	Taxes	Amount
1	Product001	1,000	£ 60,00	ST11	£ 60,00
2	Product002	1,000	£ 70,00	ST11	£ 70,00
3	Product003	1,000	£ 62,47	ST11	£ 62,47

Item	Taxes	Amount
1	Standard rate sales (20%)	£ 38,49

Total without taxes £ 192,47

Taxes £ 38,49

Total £ 230,96

Amount Due £ 230,96

Payment Status [Open](#)

Payment term: 30 Net Days

Shipping terms: FREE CARRIER

Figure 2.1.20 Repetition can be used as a positive principle of visual data communication.

In relation to data, the concept of hierarchy can also be helpful in representing relationships between levels of information. For example, in a solution that details a student and their subjects and achievements, key student details could appear at the first level, subject details at the second level, then result data at the third level. By mapping the presentation of data in a hierarchical way, the developer helps the user to interpret the data and potentially improves the way in which it is accessed, manipulated and improved.



Figure 2.1.21 Using the principle of hierarchy to allow users to interpret the levels of data relationships

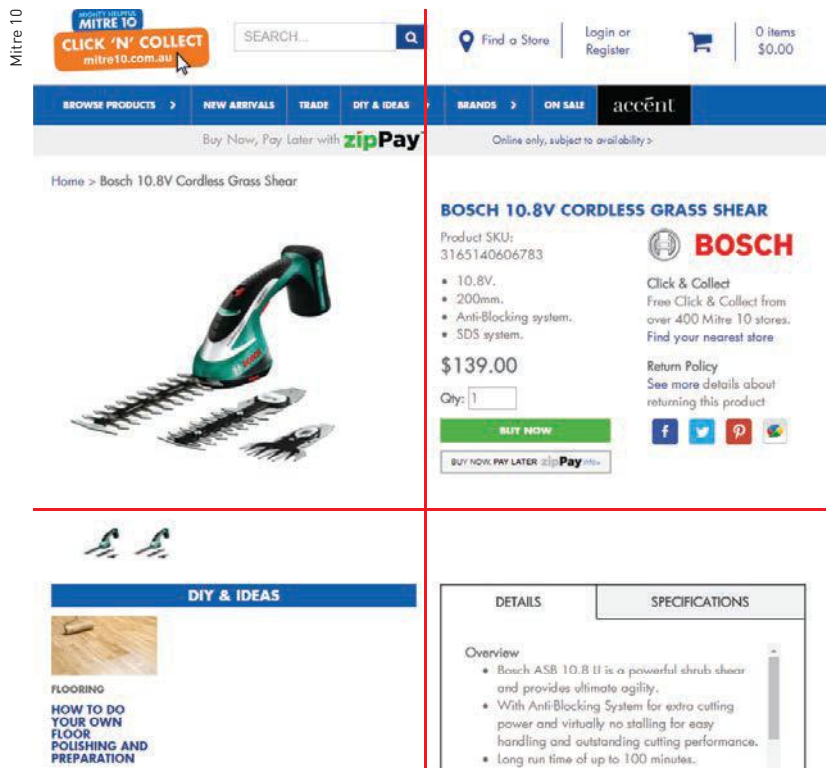


Figure 2.1.22 The principle of balance is used when arranging images and icons as well as title and detailed data. Elements of space, colour and shape are used to support this principle.

Balance

Balance in an interface refers to a sense of equilibrium and symmetry in the eyes of the user, allowing the user to effortlessly interpret the interface. Balance is sometimes informal – meaning it isn't exact, and is more of a general appreciation of perspective. Balance in a web context may mean simply that the breakdown of the page shows symmetry in relation to chunks of detail. The use of elements such as shape and space can assist with this.

Proximity

When presented with information, users will mentally group together elements that are close to each other in a space. It is suggested that many users 'clump' these into a summarised object. It is important to keep this in mind when developing a data-rich user experience.

When data is placed close to other data, a relationship is often inferred based on a user's understanding of the logical layout of data.

DATA1	DATA2	DATA3	DATA4	DATA5	DATA6
A1	A2	A3	A4	A5	A6
B1	B2	B3	B4	B5	B6
C1	C2	C3	C4	C5	C6
D1	D2	D3	D4	D5	D6
E1	E2	E3	E4	E5	E6

Changing the physical proximity of the data changes its potential meaning and how a user interprets it. This is an important consideration in delivering meaningful data to a user.

DATA1	DATA2
A1	A2
B1	B2
C1	C2
D1	D2
E1	E2

DATA3	DATA4
A3	A4
B3	B4
C3	C4
D3	D4
E3	E4

DATA5	DATA6
A5	A6
B5	B6
C5	C6
D5	D6
E5	E6

When developing your data-driven interface, you may wish to take advantage of proximity laws to enhance the users' experience and to communicate better. Examples of this are navigation controls, menu items and footer summaries on each web page. The same applies in other contexts, such as mobile applications and multimedia interfaces.

Harmony

Harmony offers an interpretation of proximity to ensure components as a whole provide valuable meaning and are complementary across the interface. In a data-driven context, it is not always harmonious to place certain datasets with others. Sometimes it is better to place datasets on separate screens to avoid confusion or otherwise improve the experience.

In a purely visual sense, using different elements in non-harmonious ways could have an inadvertent effect on the user experience. Contrasting colours, for example, if used effectively, can make a significant difference to the aesthetics of a user experience.

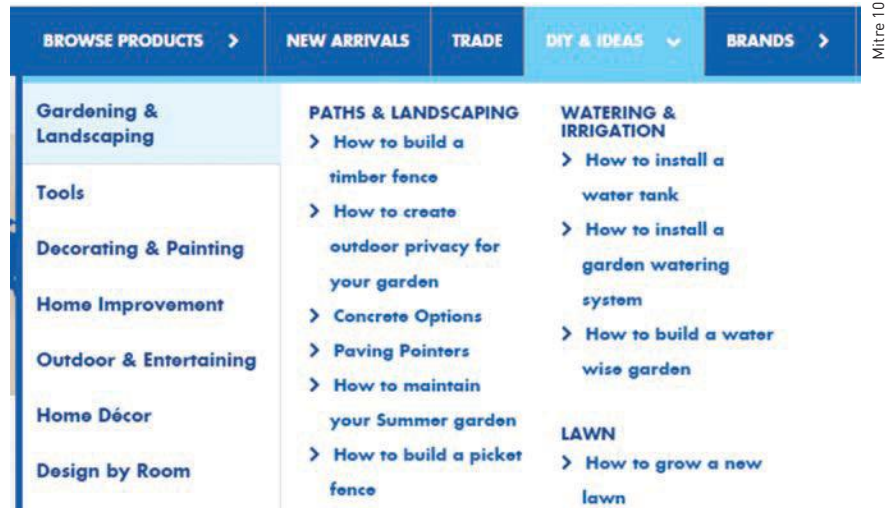


Figure 2.1.23 The menu is accessible and fixed even as users scroll. The sub-menus link and work from left to right in proximity to each other. Elements of colour, form and line work together to provide a useful user experience.

ACTIVITY 2.1.4

- 1 Hierarchical data: Identify how these three datasets could be structured on a web page for users to interpret correctly. An example might be to show all available details about the Sales department.

Workers

WORKER ID	FIRST NAME	LAST NAME	DEPARTMENT
1	John	Smith	Accounts
2	Jane	Jones	Sales
3	Julie	Abel	Accounts
4	Kathy	Timms	Sales
5	Kylie	Johns	Marketing

Departments

DEPARTMENT	LOCATION	CONTACT DETAILS
Sales	Smith St	Ext 997
Accounts	Main St	Ext 998
Marketing	Main St	Ext 999

Managers

MANAGERS	DEPARTMENT	CONTACT DETAILS
Rupert Elms	Sales	Ext 991
Ryan Barns	Accounts	Ext 992
Sally White	Marketing	Ext 993
Jessica Laws	Accounts	Ext 994
Gary Pike	Accounts	Ext 995
Andrew Lester	Marketing	Ext 996

- 2 Review an Australian chain store website. Collect screenshots of examples of visual communication principles and elements. Annotate your screenshots. Compare your results with your peers.

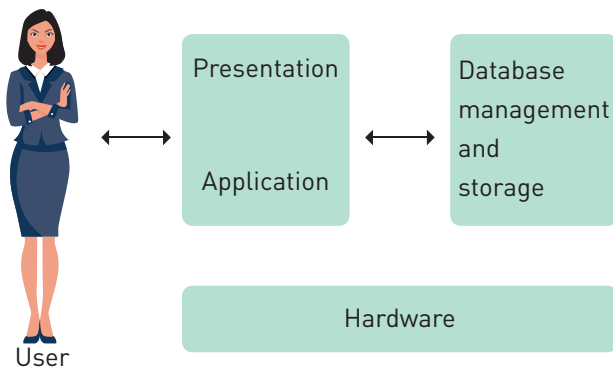


Figure 2.1.24 A simplified representation of the components that make up a typical data-driven solution

Exploring data-driven solutions

Components across systems

Typically, a data-driven solution is not just one component and is not localised to just one system. In many situations, a data-driven solution will contain application and interfacing components, data storage and management components, as well as underlying support components such as operating systems, firmware or hardware. These components work together to solve the data-driven problem.

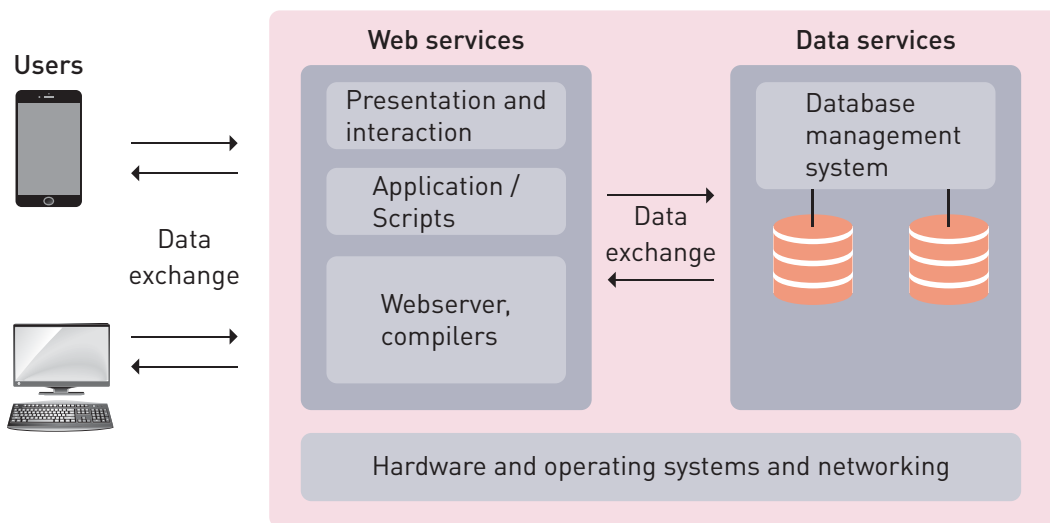


Figure 2.1.25 Web context – components of an example data-driven solution across systems

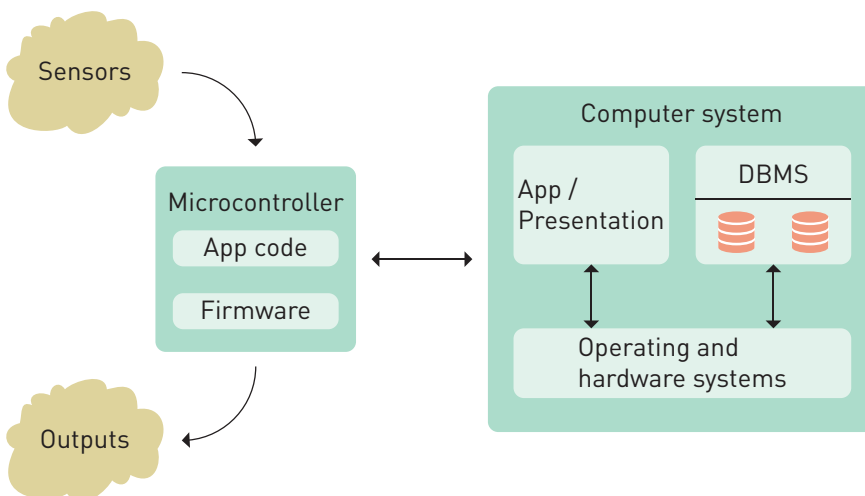


Figure 2.1.26 Intelligent system context – components of an example data-driven solution across systems

In a web context, components will exist across multiple systems.

In intelligent systems, microcontrollers may operate independently to collect, process, store and output data, but may be combined with other interfacing systems to further present and interact with data.

In the following section, the different components of data-driven solutions will be explored to gain further insight into their inner workings.

Applications, presentation and interactions

Application, presentation and interactive components are primarily concerned with the user interface or front end, input and outputs, and with the processing of any data. This is usually where a developer will generate algorithmic coded solutions and user interfaces to meet the requirements of the problem.

In a web or mobile context, this may be in a scripting programming environment such as ASP.NET using C# or VB languages and associated libraries, PHP, Javascript or Web Python. In a mobile app context, this may involve platform-specific tools such as XCode using Swift or Objective-C programming languages, Android Studio in conjunction with Java, or Xamarin using C#.

For an interactive media context, this might involve using environments such as Blender with the Python **application programming interface (API)**, or Adobe Animate with the embedded ActionScript or JavaScript programming language.

For the intelligent systems context, a mix of a number of programming environments, including Robot C, C-based variants and other compatible scripting languages that interact with microcontrollers, sensors and actuators, may be used. In this context, there also may be auxiliary interfaces based on operating system-specific platforms such as .NET or Python, or potentially other variants noted in previous contexts.

What is common amongst these contexts is the role these components play in communicating with the data system(s). In the case of commonly used relational information systems, structured query language calls will be made via a connector library and batched data will be selected, updated, inserted or deleted. Some information systems will also permit calling internally developed stored procedures, whereby the SQL instructions and exposure of the underlying database are minimised.

Data management and storage

Figures 2.1.24, 2.1.25 and 2.1.26 all illustrate data management and storage components working together to manage structured data stored in databases. Data management and storage components are often referred to as an information system. An information system generally consists of two key components:

- an instance of a database containing the data constructs, data instances and associated rules
- a database management system for managing the accessibility and storage of the data.

These two components are both necessary to store structured data and to avoid data management issues (these will be explored in later sections).

It should be noted that different forms of information systems do exist. In Digital Solutions, we primarily explore relational databases and their associated database management systems. Below is a listing of some of the different forms of information systems.

HIERARCHICAL	Hierarchical information systems were one of the earliest forms to be established, and the logic behind this model is visible in most operating systems in relation to file storage – in Windows, for example, a folder or directory tree allowing storage of files at end-node level. In a hierarchical model, every node stems from the root, and a node (folder or directory) can coexist next to other nodes and have many underlying child nodes. A node can only have one parent node. These types of information systems were widely used before other forms. Algorithms such as tree-spanning algorithms were developed to support such systems, and these algorithms are still used today in different and evolved contexts.
NETWORK	Network information systems were a slight evolution on the hierarchical versions as they did not have the 'single parent node rule'. This was invented to reduce the duplication of data that needed to exist in separate branches of the hierarchical tree structure. Nodes were interconnected to multiple parent nodes. This form of information system, while useful, is considered by many to be complex to work with.
OBJECT-ORIENTED OR OBJECT	Object-oriented information systems follow principles similar to those found in object-oriented programming, including classes, instantiation of classes as objects, object inheritance, properties and methods. These systems contain data and processes to work with the data. These databases have continued to gain momentum in niche situations, including complex engineering and spatial data.
RELATIONAL	The relational model is the most widely recognised and adopted model. They can be well structured and have minimal problems associated with redundancy and integrity if designed correctly.
FLAT FILE	A flat file database is a simplistic data model whereby all data is contained in a single file (such as a text file), a downloadable data file, or on a single spreadsheet (table). A flat file can support simplistic data storage and manipulation but significant problems can occur once higher levels of data complexity are reached.

Flat files as databases

Some databases are relatively simple in nature and only need a single table to store information. This is known as a flat file. These types of data stores can exist in the most primitive forms, such as an administered text file or a spreadsheet, or can be constructed within various database management tools.

STUDENT ID	LAST NAME	FIRST NAME	ADDRESS	AGE	SUBJECT
1	Smith	Sally	25 Koola St	13	English
2	Jones	Peter	123 Factory Rd	12	Maths
3	Zen	Adrian	9 Acacia Ave	14	English
3	Zen	Adrian	9 Acacia Ave	13	Science

While a flat file as a database is easy to use and useful in managing small, simplistic datasets, problems can occur when they grow or when additional data needs to be included in a record (row). As shown in the example flat file table above, a second subject entry is needed for student number 3. The only way to enter this information is to re-enter the entire record. This poses an issue in that the same data must be duplicated, causing unnecessary replication of records. The information within the new record contains redundant data. Removing **data redundancy** from databases is of primary concern to database designers as it can lead to other problems, such as data integrity issues. An example of a data integrity issue is shown in the ages entered above for student 3. In one record, a user has entered '14' as the age. In a second record, a different user has entered '13' as the age. Databases are generally designed to be accessed by multiple users, and other users will now have to question which age for student 3 is correct as a discrepancy now exists. From a user's perspective, there is now a reliability issue with the database. The reliability of a database is essential to its ongoing use, and database designers and administrators must mitigate as many of these issues as possible at the design stage. In many cases, database designers use other database models, including relational databases, to reduce these issues.

Relational databases

The relational model for databases was first devised by E.F. Codd in the early 1970s and is based around the notion of a relation. It logically represents a database as a collection of relations, and these are represented in the form of a table of values. Data is stored in a table across columns and rows. Each column represents a particular field, and these are given data content-descriptive names. In many systems they are given a data type that can be stored in that field. Each row in a given table is described as a record. A record is an interlinked series of field values that together make up a meaningful set of collated data, or information. Using a school database as an example, a table of records might look something like this.

STUDENT_ID	FIRST_NAME	LAST_NAME	DATE_OF_BIRTH	GENDER*	YEAR_LEVEL	ENROLMENT_DATE
00001	Amelia	Thomson	09/11/2005	F	9	30/1/2016
00002	Adelaide	Wilson	20/04/2007	F	7	27/1/2019
00003‡	Roy	See	07/12/2003	M	11	6/7/2013
00004	Doug	Butler	04/02/2004	M	10	4/3/2014**
00005	Noel	Heath	01/06/2002	M	12	01/02/2012
00006	David	MacDonald†	22/09/2001	M	12	03/02/2013
00007	Thomas	James	18/10/2006	M	8	30/1/2016
00008	Adelaide	Wilson	11/07/2004	F	10	31/1/2016

Legend: * Field, ** Record (entire row), † Data value/instance, ‡ Key

The concept of relations is the cornerstone of a relational database. The success of this type of database is forged on the effective use of keys and applying constraints. These will be explored in later sections.

Keys

If you look carefully at the table in the previous section, you will notice that there is a `Student_ID` field in each row. You may also note that it is always different in each record. This is because the `Student_ID` makes each record unique. Why is this important? Names are generally not unique. A name only has to be typed into a search engine to see that there are hundreds, if not thousands, of people with the same name or one similar. In the case of the example dataset, there appear to be two students with the name 'Adelaide Wilson'. They are in different year levels, have different birthdays and entered the school at different times. However, if we relied purely on their names to make their record distinguishable from others, the database would ultimately fail. This is why alphanumeric identifiers are widely used in relational databases. This process of being given a unique identifier is something we now take for granted. Think of mobile telephone numbers, bank account numbers, Medicare numbers, product barcodes and book ISBNs – these are all ways in which entities are uniquely identified.

In the case of relational databases, keys are taken a step further in that these form the method of interconnecting tables within the database. One of the significant advantages of spreading data records across multiple tables is that it improves the total efficiency of the database, removes redundant data and mitigates a range of data issues that could potentially occur. These potential issues will be explored in detail later in this section.

Let's extend the previous example to understand the relationship of keys. In this version, we now have three tables. The original table still contains the student core information. The second table contains behaviour records entered by teachers. The third table contains information about subjects and overall results for a given year.

Table 2.1.2 Student information table

STUDENT_ID	FIRST_NAME	LAST_NAME	DATE_OF_BIRTH	GENDER	YEAR_LEVEL	ENROLMENT_DATE
00001	Amelia	Thomson	09/11/2005	F	9	30/1/2016
00002	Adelaide	Wilson	20/04/2007	F	7	27/1/2019
00003	Roy	See	07/12/2003	M	11	6/7/2013
00004	Doug	Butler	04/02/2004	M	10	4/3/2014
00005	Noel	Heath	01/06/2002	M	12	01/02/2012
00006	David	MacDonald	22/09/2001	M	12	03/02/2013
00007	Thomas	James	18/10/2006	M	8	30/1/2016
00008	Adelaide	Wilson	11/07/2004	F	10	31/1/2016

Table 2.1.3 Student behaviour table

BEHAVIOUR_RECORD_ID	COMMENT_DATE	STUDENT_ID	COMMENT	ENTERED_BY
1	23/4/2019	00003	Roy was talking excessively in his English class and had to remain after class to finish his work.	TCHAB01
2	24/4/2019	00005	Noel was given a detention for being 10 minutes late to class. He had no explanation for why he was late.	TCHQT56
3	24/4/2019	00007	Thomas left rubbish around his seating area at lunch. He spent the next break picking up rubbish from the area.	TCHAR43
4	25/4/2019	00003	Roy did not complete his homework for English for a second time. Parents have been contacted via email.	TCHAB01

Table 2.1.4 Student subject results table

SCHOOL_YEAR	STUDENT_ID	SUBJECT_CODE	REPORT_COMMENT	RESULT
2018	00003	ENGLISH11	Roy has been progressing well in English ...	B+
2018	00003	MATHA11	It has been a challenging year in Maths A for Roy ...	C+
2017	00003	ENGLISH10	Roy has been developing his writing skills throughout Year 10 in English ...	B
2018	00005	ENGLISH12	Noel has worked incredibly hard this year in English and should be commended for his efforts ...	A

The keys now act as links between the tables and allow for related data to be stored and later manipulated into useful information in the external view of an application.

At this point we can make distinctions between the types of keys that can exist in a well-designed relational database. A **primary key** is the unique value in each record that relates to all other data fields for that record. So, in the student details table, this is the Student_ID as it is totally unique in every record and relates independently to all data values across the fields in a given record.

In the student behaviour table, it is the Behaviour_record_ID that is the unique identifier. You can determine this by noticing that the values are unique in each record and relate again to each other field value. It is not the Student_ID that is the unique value in this table because the same student may have multiple behaviour records.

The third table relating to student subject results is a little more complicated. As hinted at earlier, it is possible to have multiple primary keys in a table. What this means is that multiple keys must work in conjunction with each other to establish uniqueness across each record. This situation is commonly referred to as **composite keys**. In the case of the student subject results table, there are three primary keys working together to give each record distinction from each other. The School_year, Student_ID and Subject_code identifiers are together forming a composite key that is never repeated.

The Student_ID appears a few times in this example. In the student information table, it is purely a primary key. This is its base table; where it is initially created before it can appear elsewhere in the database. In the other two tables it is used to link to the student table. In this case it is referred to as a **foreign key**.

Foreign keys are those data records in another table that relate to the current record. They prevent unnecessary duplication of data and can be used to look up data in the base table. In the student database example, the Student_ID is included as a linking field in the student behaviour table. It may appear more than once, as it is not controlling the uniqueness of the records. It refers back to the student information table and adds meaning to the behaviour record. Even though we can see student's names listed in the teacher comments, it is the Student_ID field that links back to the student information table and identifies the student correctly. What if a teacher types the wrong name in the comment? A user error of this type is unfortunate but, importantly, it does not corrupt the rest of the data in the database. The integrity and reliability of the database is maintained by the quality of the relational design.

Uniqueness constraints

In a relational model, uniqueness is an interrelating constraint between tables. It is an important condition that should be understood so that the database can be used effectively. The constraints themselves act as rules to guard the integrity of the database and to ensure its continued viable use.

Let's review the previous school database example now in the form of a table relationship diagram.

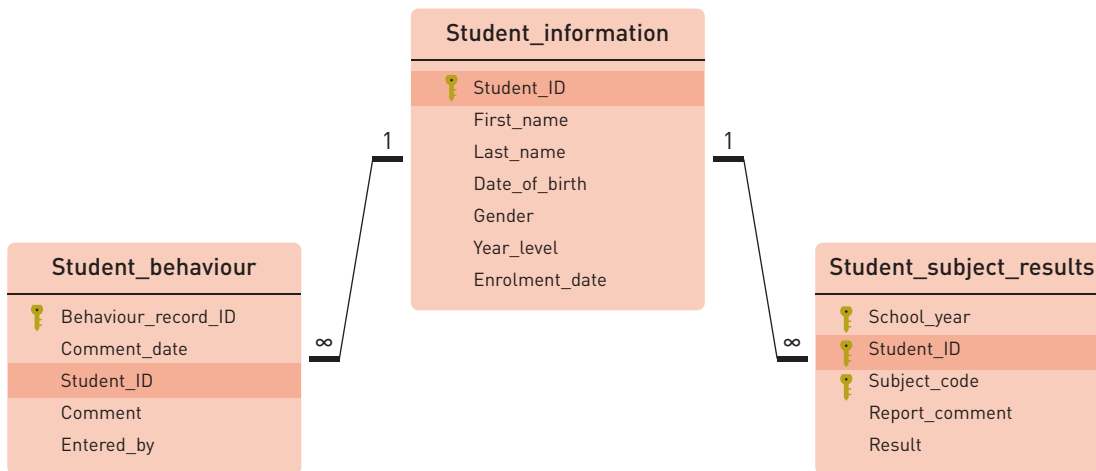


Figure 2.1.27 Links between primary and foreign keys

The diagram illustrates the links between the primary and foreign keys of each table. It also shows the primary keys, which are denoted with a key character. The uniqueness constraints applied in this example are noted at each end of the lines as '1' and ' ∞ ', which represent 'one is to many'. What this means is that for a single **Student_ID** unique value in the student information table, there can be between none and any possible number of duplications of that value in the student behaviour table. Conversely, for a single **Student_ID** unique value in the student information table, there can also be between none and any possible number of duplications of that value in the student subject results table.

These constraints ensure the quality of the database values and mean that invalid values (for example, ones that don't exist in the student information table) cannot be committed to tables.

There are various options of **uniqueness constraints** that are applied in a relational database. Following is a summary of the different types.

One-to-one relationship

A one-to-one relationship can exist between two tables in which a single unique value in one table must only exist up to once in the second table. As a constraint, it is a rare occurrence. One possible scenario is when there are multiple unique identifiers that require separation between tables.

One-to-many relationship

As defined in the previous example, a one-to-many relationship can exist between two tables in which a single unique value in one table can exist between zero and any number of times in the other table. An example of this in the earlier Garden Express web store case study might be that a customer can exist against multiple orders. In reverse, a customer listed against multiple orders can only exist once as a customer.

Many-to-many relationship

A many-to-many relationship can exist between two tables in which a unique value can appear multiple times in the first table, and also appear multiple times in a second table. Generally, this is rare as a uniqueness constraint between tables. It tends to imply a need for composite keys inside a table, which is determined through what is called the **normalisation** process.

The process of determining **cardinality** (one to one, one to many, many to many) of uniqueness will be examined later in section 2.2.



Database field data types

As in programming storage, databases generally require the identification of data types to accept as a stored field value in a table record. This is because storage and handling allocations must be created to cope with the specific values. The database management system can also interpret certain operations accordingly and follow rules and practices. Each database management system has specific data types that will need to be explored further. Following is a core guide to common data types within databases.

Integer

An integer is essentially a signed or unsigned whole number without a decimal point. In most databases, there are variations of integer data types that make allowances for very small ('short') to intermediate (usually just referred to as 'integer') and very large ('long') ranges. The selection of the integer data type should be determined by the nature of the value and how big or small it will get. For example, it would be wasteful to select 'long' as a data type for a field holding a value representing human ages, as few humans live beyond about 100 years of age.

Float

A float is a generic name for a number with a floating point within it. In more generic mathematical terms, it has a decimal point and is not purely a whole number. Depending on the relative size of the value, options may include 'float', 'real', 'decimal' and 'money'. Larger decimals with exponents may require a specific data type, compared to the simple two decimal places required for monetary values.

Date and time

A common requirement in any database is to record date and time values. For example, a person's date of birth or the date on which a sale takes place. Strictly speaking, date and time data types have been built on top of primitive numerical data types for ease of use. Most database management systems record a date in a universal format, from largest to smallest. For example, 11.23 p.m. on 23 April 2010 would be recorded as '2010-04-23 23:23:00.00'. It is worth noting in this example that the p.m. time is saved as a 24-hour time that also includes seconds and fractional seconds. Applications are then left to present the data in a style that suits the user. In some database management systems, you may choose a data type that simply stores the date without time; conversely, you can store the time without the date. As with previous data type groupings, careful consideration must be given to selecting the right data type for the job with the smallest amount of storage required.

Boolean

One of the simplest data types, Boolean data is stored as one of two options: true or false, on or off, yes or no. Some database management systems explicitly provide a true Boolean data type. Others, such as Microsoft SQL Server, essentially use other data types for storing representations of this. It is worth investigating your chosen database management system for Boolean data types before applying them.

ACTIVITY 2.1.5

- 1 In Microsoft SQL Server, what is the difference between char, varchar, nchar, and nvarchar data types? Why are there so many variations?
- 2 Why do text/ntext data types exist in some database management systems? Are they efficient in using storage? When should they be used?

Characters and strings

Characters and strings of characters (usually referred to as text) are widely used in all database management systems. Strings are essentially an extension of character data types that are linked together behind the scenes. In many database management systems, 'char' data types require a numerical setting to determine how many characters can

be stored in the value. It is worth noting that some of these allocate storage for the theoretical maximum for every value stored, while others (such as 'varchar') are more dynamic and can potentially reduce the storage required.

Table 2.1.5 The most common data types used in databases

DATA TYPE	EXAMPLE	EXAMPLE USAGE
Characters and text	In Microsoft SQL Server, 'varchar(50)' represents a string of characters up to a maximum of 50.	A field holding the last name value of a person may be set to varchar(100).
Integer	In PostgreSQL, 'smallint' is 2 bytes in storage size and holds values between -32768 and 32767.	A field holding a person's age may be set to smallint.
Date and time	In Microsoft SQL Server, 'datetime' represents a value from year down to microseconds.	A wave collection table may contain a field of when the height of waves at a specific location was collected, e.g. '2018-10-10 13:22:00.03'.
Float	In MySQL, 'Decimal' can be used and the number of values before and after the decimal point can be set.	A field for an employee salary might be set to Decimal(6,2). An example value is 126850.88.
Boolean	In PostgreSQL, 'Boolean' field records 1 or 0.	A field called 'smoker' records 1 or 0.

PostgreSQL database management system data types

Table 2.1.6 Integer/whole number data types

NAME	STORAGE SIZE	DESCRIPTION	RANGE
smallint	2 bytes	Small-range integer	-32768 to +32767
integer	4 bytes	Typical choice for integer	-2147483648 to +2147483647
bigint	8 bytes	Large-range integer	-9223372036854775808 to 9223372036854775807
serial	4 bytes	Auto-incrementing integer	1 to 2147483647
bigserial	8 bytes	Large auto-incrementing integer	1 to 9223372036854775807

Table 2.1.7 Float data types

NAME	STORAGE SIZE	DESCRIPTION	RANGE
decimal	Variable	User-specified precision, exact	Up to 131072 digits before the decimal point; up to 16383 digits after the decimal point
numeric	Variable	User-specified precision, exact	Up to 131072 digits before the decimal point; up to 16383 digits after the decimal point
real	4 bytes	Variable-precision, inexact	6 decimal digits precision
double precision	8 bytes	Variable-precision, inexact	15 decimal digits precision
money	8 bytes	Fixed fractional precision	-92233720368547758.08 to +92233720368547758.07

Table 2.1.8 Character data types

NAME	DESCRIPTION
character varying(n), varchar(n)	Variable-length with limit
character(n), char(n)	Fixed-length, blank padded
text	Variable, unlimited length

Table 2.1.9 Boolean data types

NAME	STORAGE SIZE	DESCRIPTION
Boolean	1 byte	Value is set to 1 or 0 representing true or false

Table 2.1.10 Date and time data types

NAME	STORAGE SIZE	DESCRIPTION	RANGE
timestamp	8 bytes	Both date and time	4713 BC to 294276 AD
date	4 bytes	Date (no time of day)	4713 BC to 5874897 AD
time	8 bytes	Time of day (no date)	0:00:00.00 to 24:00:00.00
interval [fields]	16 bytes	Time interval	–178000000 to 178000000 years

Note: Variations exist with possible inclusions of worldwide time zones when necessary.
[fields] represent YEAR, MONTH, DAY, HOURS, MINUTES, SECONDS.

Selecting an incorrect data type can cause long-term issues that include excessive storage and, potentially, speed. Databases are typically stored in secondary memory such as hard drives, and hence are much slower than data stored as variables in programs temporarily held in primary memory. As databases grow larger, they require additional storage, and the read time for a database management system to interrogate and manipulate data can grow. This can be exacerbated by selecting data types with storage allocations that are larger than required. It is good practice to limit the data type requirements to the needs of the field. But this must be weighed up against future growth in values. A good example of allowing future growth is enabling a larger integer field to house larger indexing numbers as time progresses.



Issues of data management

There are a number of issues that must be considered when managing data, particularly when it is being collated and manipulated within a relational information system. In this section, we will explore and unpack some of the key issues relating to data management and how you can prepare for them in the development and generation of your data-driven solutions.

Acquisition

Data acquisition refers to the collection of data. Although this sounds simple, there are a number of potential issues that need to be considered. Data that can be acquired may come from many sources – amassed in digital forms throughout the world wide web, within digital documents, on organisational servers, intranets and in paper form in various physically stored systems. To unpack the potential issues faced in data acquisition depends on the scenario and context of the data being acquired, what it is being used for and how it needs to be stored. In this brief review of data acquisition, we will highlight a number of issues faced.

Time

Within any data process, the collection of data will be delayed by time, processes or people. The issue of delays in data being acquired into a system can play havoc with an organisation's system.

For example, manual entry of form data can potentially slow the input of data into a system. Paper-based forms may require entry by one department of an organisation. Other areas of that organisation may need to wait before they can continue completing any processes. An example of this may be when someone begins working for a large employer. The department hiring the person may negotiate them to start on the first day of the month. The employee completes all of the necessary forms and submits them to the human resources department as requested.

This department only has two staff members, one of which is on leave. There has been a larger than usual turnover of staff across the organisation and there is very large number of new staff processes to be completed. The manual entry of data is seen by the Human Resources staff as reason for data problems in the organisation.

Some organisations use form-processing technologies such as optical character recognition (OCR) to scan paperwork and automatically harvest information. The advantages of using OCR technology-ready data-collection forms are that there is no requirement for users to have digital access, and that paper-based documentation is easily accessible. The disadvantage is that OCR is not 100 per cent accurate and minor errors can occur. While this small percentage of paperwork must still be handled manually, OCR has the capacity to improve the process of data acquisition in a timely manner. OCR-ready forms usually contain characteristics such as clear data elements, individual character entry boxes for letters, checkboxes, and clear zones for signatures.



Ownership

Another issue related to data acquisition is the ownership of the data. Some data is held in relation to clients and may be sensitive in nature. A hospital with patient data is a prime example. The hospital would collect initial information about the patient, including past medical information. They would further add to their patient records data relating to procedures and medical services completed at the hospital by medical staff. This information is technically owned by the patient and is collected according to stated privacy policies and procedures.

Other systems may source information that is officially owned by other individuals, organisations and governments. This could be data ‘scraped’ from different online sources such as public web pages, publicly accessible networked documents and emails, or data collected from digital publications, open datasets or RSS feeds. Regardless of the method of data collection or the type of usage, the ownership and licensing of data must be considered, particularly if this information is being re-presented or is being on-sold for commercial gain.

ACTIVITY 2.1.6

Investigate the characteristics needed for OCR-ready forms.

Using an existing school enrolment or data-gathering form, redesign the form into a OCR-ready template. When completed, describe the key features of your OCR-compliant form.

Data integrity

When terms such as integrity are used in relation to information systems, they refer to the quality and consistency of the data. The integrity of a database depends on it being complete, accurate, up-to-date and secure, and on maintaining relevancy. The integrity of the database should remain a high priority for its entire life cycle. It is essential that the data is valued and can be trusted by users. If the data within a database was to become questionable in some way, if it became ambiguous or lacking distinguishable meaning, the database would eventually be retired from use. Critical historical information stored over time could also be lost due to a lack of integrity.

Integrity issues can arise in many ways, but the most obvious way is through design flaws in the database itself. Let’s investigate a potential data integrity issue using the school database scenario.

ACTIVITY 2.1.7

Review the two following scenarios in class and discuss whether a data ownership and licensing issue is at stake.

- Scenario 1: A marketing group has built a data-driven system that continually scrapes key data elements from a number of advertising sites to gather insights into product placement and digital advertising. They process this data to develop summaries, and store the final results in their system for analysis and use.
- Scenario 2: James is building a data-driven application that is collecting news stories that appear via limited-life news feeds from across Australia. He tabulates these stories and stores them in a data-driven news archive system that he intends to offer as a subscription service to users.

Table 2.1.11 Student information table

STUDENT_ID	FIRST_NAME	LAST_NAME	DATE_OF_BIRTH	GENDER	YEAR_LEVEL	SPORTING_LEVEL
00001	Amelia	Thomson	09/11/2005	F	9	Under 14s Girls
00002	Adelaide	Wilson	20/04/2007	F	7	Under 12s Girls
00003	Roy	See	07/12/2003	M	11	Under 16s Boys
00004	Doug	Butler	04/02/2004	M	10	Under 16s Boys
00005	Noel	Heath	01/06/2002	M	12	Open Boys
00006	David	MacDonald	22/09/2001	M	12	Open Boys
00007	Thomas	James	18/10/2006	M	8	Under 14s Boys
00008	Adelaide	Wilson	11/07/2004	F	10	Under 16s Girls

In this example, a sporting group has been allocated in every record. This grouping is dependent on the age that the student is reaching in the given year and their gender. A year passes and student records are manually updated. While some records have been correctly updated, other records have not.

STUDENT_ID	FIRST_NAME	LAST_NAME	DATE_OF_BIRTH	GENDER	YEAR_LEVEL	SPORTING_LEVEL
00001	Amelia	Thomson	09/11/2005	F	10	Under 16s Girls
00002	Adelaide	Wilson	20/04/2007	F	8	Under 12s Girls
00003	Roy	See	07/12/2003	M	12	Open Girls
00004	Doug	Butler	04/02/2004	M	11	Under 16s Boys
00007	Thomas	James	18/10/2006	M	9	Under 14s Boys
00008	Adelaide	Wilson	11/07/2004	F	10	Under 16s Girls

In the case of student '00002' the sporting level has not been updated; student '00003' has been incorrectly attributed to a new group (Open Girls – should be Open Boys); and the year level for student '00008' was not updated. This may be for a valid reason – for example, if the student is repeating the year. But, as trust in the data diminishes, it may not be considered accurate enough to depend on.

Integrity of a data-driven system ultimately depends on data being:

- complete – all data elements are entered correctly and are not missing elements
- accurate – the data represents the 'truth' and can be trusted and relied upon
- kept up to date – all data and changes to data occur in a timely manner
- consistent – the manipulation of data is completed in a consistent manner. For example, updates to year levels could be achieved via a triggered, scripted UPDATE application feature, rather than manually updated by users with the interface
- secure – the security of the system is paramount in order to protect the data from others who may intentionally corrupt it
- relevant – at the design stage and throughout usage, the relevancy of the data to the intended universe of discourse is maintained. In other words, the data stored has meaning and use in the context of the system.

Data anomalies

Data anomalies are potential problems caused when modifying data in databases that have not been developed well enough and contain redundant data. There are three types of data anomalies: update, insert and delete.

An update anomaly is caused by inconsistent data due to the existence of redundant (replicated) data, and by the database being only partially updated. For example, each student at the school is in a year level as well as a sporting team.

STUDENT_ID	NAME	YEAR	TEAM
876	Jill	10	FSSL
456	Adrian	11	Carry Ons
456	Adrian	11	Wanderers
734	John	12	Sprints
734	John	12	Wanderers
845	Lily	10	Scorchers

If John's year level was initially entered incorrectly, the error must be updated at least twice or there will be inconsistent data in the database. If the user performing the update is not aware of this, only a partial update will occur. This means that the update was not done properly and there will be an update anomaly in the database that makes it look as if John is in two year levels.

A deletion anomaly is caused by the unintended loss of data due to deletion of other data. For example, if the team 'FSSL' was disbanded and a user decided to delete it from the table, Jill's record in the database would no longer exist. A deletion anomaly has caused her record to be unintentionally removed.

An insertion anomaly is caused when a record cannot be added to the database because of the absence of other data. For example, let's assume that the 'Team' field must always contain a value (no null values are allowed). If a new student enters the school but does not initially join a sporting team, then their student record cannot be entered into the database. An insertion anomaly occurs in the database due to the omission of data.

Overall, anomalies are something that should be avoided when developing or maintaining a database. Anomalies can be avoided by following processes such as normalisation to remove redundant data.

Security and protection

Security is essential to the success of any information system and the valuable data stored within it. Security is a necessity given the variety of threats that exist in our digital world. The potential threats include the following.

Malware

Malware is software with malicious intent that can infect systems that serve a data-driven application. Malware examples include computer viruses, worms, Trojan horses, spyware and adware. Malware can support or commit harmful actions such as stealing sensitive information or corrupting it in some way.

Data theft and identity theft

Data theft and identity theft are major concerns in relation to information systems and data-driven applications. Systems whose security measures become compromised can be mined for valuable, sensitive and private data. In the case of identity theft, this could include identification details, bank account details and other highly sensitive information. Identity theft is of major concern as individuals may pretend to be others in order to steal money, or commit fraud or other crimes.

Invasion of privacy

Most countries and jurisdictions around the world have enacted legislation requiring governments and companies to protect an individual's personal information that it might store and use. Issues surrounding the security of data access and transmission can lead to invasions of privacy as information may be accessed, leaked, gathered, and used to blackmail or for unsolicited marketing.

Data management security solutions

User-level access

The use of user-level access privileges provides a way for data administrators to secure information from unauthorised users and only give access to those who need it. This can be achieved by requiring username and password credentials and aligning access to application features when authenticated. For example, until a user is authenticated, they cannot access certain data-driven web pages within a site.

Encryption

Another method for protecting and securing data, databases and the overall data-driven solution is the use of encryption processes. We will explore encryption in two parts: data storage encryption processes and transmission.

Some data that is stored within a database can be highly sensitive and need protection. This can include special security levels or user passwords, and these must exist in some form that allows users to be challenged and allows them to enter passwords to substantiate and authenticate their access rights and privileges within the system. Keeping a password in plain text within a data store is highly risky and could compromise the security of the system. This is why it is generally accepted practice that passwords are encrypted or 'hashed' so that they cannot be read in plain text. The distinction between encryption and hashing is that encrypted data by definition can be decrypted using a key, while hashing produces an indistinguishable alphanumeric code series in place of the data. Hashing is a preferred method for password storage in data stores such as databases. There are various hashing algorithms, and some are considered best practice due to the extreme difficulty of breaking them. Others, such as MD5, are no longer cryptographically secure and would be avoided in real-world deployments. But, as a useful learning exercise and a method that it is still widely deployed in data-driven solutions, we will explore hashing using the MD5 algorithm.

How hashed password authentication works

Before we begin, a password must be stored in the database field. Suppose a user provides the password 'Banana43\$' and, using the MD5 hash algorithm, produces a 30-character hashed code, '51fea8c739ebfa1072d255b02f666a7e'. This is then stored in a database table.

USER ID	EMAIL	PASSWORD
1	john@thehut.com	27548d65ac3e4f068311651a0cf51a64
2	jamila@thehouse.com	1f8094bb6297edadc7755127ec8c8d1d
3	jill@thebarn.org	d167436cbf43c05a3c570729913e8d60
4	sally@theoffice.net	51fea8c739ebfa1072d255b02f666a7e

Step 1: Compare usernames

Username entered by user
sally@theoffice.net

✓
Successful
match

Username entered by user
sally@theoffice.net

Step 2: Compare passwords

Password entered by user
Banana43\$



MD5 hashed entered password
51fea8c739ebfa1072d255b02f666a7e

✓
Successful
match

Hashed password stored in database
51fea8c739ebfa1072d255b02f666a7e

Step 3: Authorise user

The user is authorised into the system. In a web/mobile web context, this usually involves activating a user session process or other similar method.

Figure 2.1.28 Steps for testing user credentials against stored data



A user then enters their user credentials into the login screen to authenticate.

These credentials are tested against the data stored in the database. But, as mentioned earlier, hashing is only one way of storing data. Therefore, before an effective comparison can be made, the password entered by the user must also be hashed. The information can then be compared.

ACTIVITY 2.1.8

Note: Not all of these activities may incorporate hashing. However, this is something that can be added in the next iteration of the challenge.



Backup and disaster recovery processes

It is essential as part of a disaster recovery plan that databases are regularly backed up. They should also be configured to log data transactions in case specific committed data changes need to be rolled back. Most database management systems provide support for automated backup processes.

Exploring relational database management systems

As previously mentioned, a database management system is a software system that employs standardised ways to store, catalogue, retrieve, modify and execute queries on data. They manage generated database structures that have been determined through analysis of particular data. They deal with incoming data, organise it in structured ways, and provide methods for it to be modified or extracted by users or other programs. Throughout this section, these methods will be investigated.

There are a number of database management systems that can be used to explore, develop, generate and evaluate data-driven solutions in this unit. Most relational database management systems contain similar functionality on the surface, but may require further investigation for specific use. Some of the more popular relational database management systems are shown in table 2.1.12.

GROUP ACTIVITY 2.1.4

In groups, find a news article regarding a hospital data scandal.

Your task is to investigate the issues of data management at play and summarise them in relation to the concepts mentioned in the preceding section. Your group should then develop a list of possible actions and changes to the aforementioned system to avoid such issues into the future.

Each group can present its findings in a class discussion where options can be explored and shared.

Table 2.1.12 Popular relational database management systems

DBMS	DESCRIPTION	RESOURCES
MySQL Server	MySQL server is a widely used database management system and includes both a community (open source) and supported editions. It can run on a variety of service platforms and is heavily used in a web/networked scenario.	MySQL Server Community Edition It is recommended to also include MySQL Workbench in a development environment to provide a visual tool to work with databases.  2.1.8 SQL
Microsoft SQL Server	Microsoft SQL server is a proprietary database management system that is widely used by industry and is used in local, networked, desktop and web scenarios. It can be scaled for small solutions all the way up to multi-site interconnected database solutions. It can run on Microsoft desktop or server operating systems and is also available within Microsoft's Azure SQL cloud computing suite of tools and services.	Microsoft's SQL Server comes in many editions and requires a proprietary licence. Full versions may be available to schools depending on Microsoft licensing arrangements. Microsoft SQL Server Express edition is free to all users and can be used successfully within Digital Solutions. It is recommended that MSSQL Management Studio is also installed with any version of MS SQL Server to provide a visual tool to work with databases. The cloud-based Azure SQL server services are a viable and contemporary alternative to localised install.  2.1.9 SQL database



DBMS	DESCRIPTION	RESOURCES
Microsoft Access	Microsoft Access is generally accessible as part of Microsoft's Office suite of productivity applications, alongside Word, PowerPoint and Excel. It is capable of managing small to intermediate databases that can be networked or work with various application layers including web. Another reason for its popularity is that Microsoft Access includes its own application layer for forms and reports. These can be arranged in menu-driven contexts, allowing for rapid application development. It is not a viable solution for large-scale implementations. Being a Microsoft tool, there is portability between Microsoft SQL server and Access.	Microsoft Access is generally part of Microsoft's Office productivity package and is widely accessible to schools depending on Microsoft licensing arrangements. It is worth noting that compatibility between applications and Microsoft Access database files is usually dependent on them being saved in the legacy '.mdb' format due to limitations of connectors.
Oracle	Oracle database management systems are generally used by large corporations and governments and are a noteworthy RDBMS. Oracle's cloud-based database tools and services are easy to explore and can be trialled for learning purposes.	 2.1.10 Oracle
PostgreSQL	PostgreSQL is a fully open-source RDBMS that can work in a wide variety of programming application environments and can be installed on a variety of platforms. It is considered one of the best database management systems for adhering to standards and is favoured by some for this reason.	 2.1.11 PostgreSQL
SQLite	SQLite is an extremely lightweight and easy database management system to use and incorporate. It has the ability to run on resource-rich hardware down to some microcontroller-based environments and linked storage. It may be a viable option for implementing the intelligent systems or mobile applications context – particularly in a non-networked solution scenario.	An extremely lightweight and easy environment to run. A number of visual editors are also available.  2.1.12 SQLite

TEST YOUR KNOWLEDGE

- Explain each of the following concepts in the data-wisdom paradigm.
 - data
 - information
 - knowledge
 - wisdom

FRIENDS OF THE LIBRARY
Membership Application

Name:

Library Card No.:

Address:

Phone No:

E-Mail:

Date of birth:

I would like to help at:

- ☐ Burrum Heads
- ☐ Hervey Bay
- ☐ Howard
- ☐ Maryborough
- ☐ Tiaro

Is this application fulfilling employment agency requirements?

Yes No

Signature:

Date:

Figure 2.1.29 Example of an online form

- Using the form shown in figure 2.1.29, identify the different data elements by giving each a descriptive name. Identify possible storage data types for each element.
- Explain the characteristics and differences between the external, logical and internal view perspectives of data.

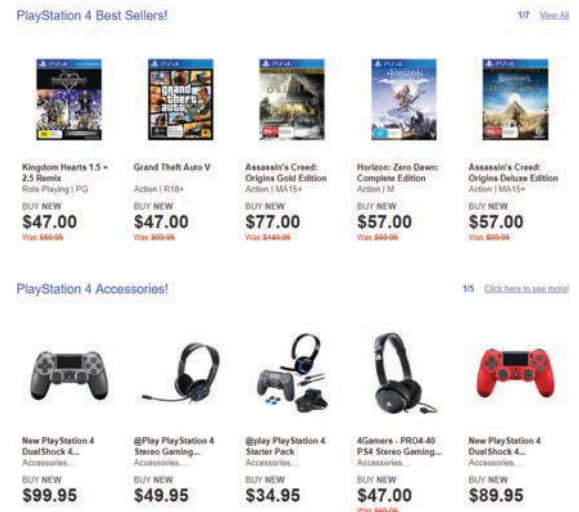


Figure 2.1.30 Example of an interface

- Using the example of the interface shown in figure 2.1.30, identify the visual communication elements at play. Describe how they have been used to support a user's interaction with the application and data.

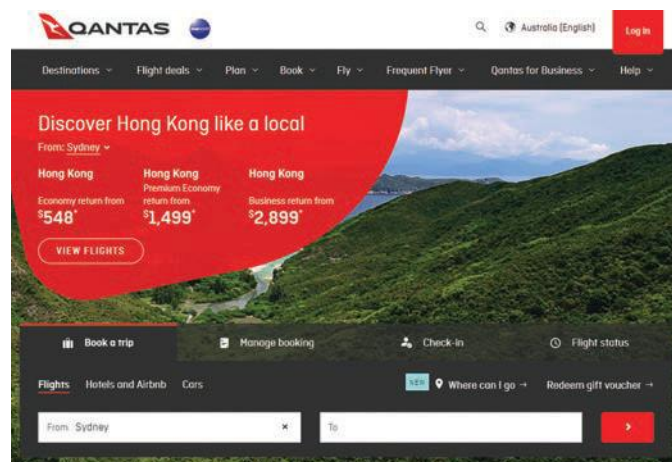


Figure 2.1.31 Example of an interface

5 Using the example of the interface shown in figure 2.1.31, identify the visual communication principles at play. Describe how they have been used to support a user's interaction with the application and data.

6 Using the sample table shown, identify the following features.

- a record
- b field
- c value

STUDENT	YEAR	RESULT	COMMENT
1	2018	A	An excellent result.
1	2019	A-	Provided excellent answers with only minor errors.
1	2020	A	Overcome minor errors to gain an excellent result.
2	2019	B	Worked hard.
2	2020	A	An excellent result. You should be proud.
3	2020	C	Is gaining understanding.

7 What is the purpose of a primary key in a data table?

8 What is SQL?

9 Using the sample table from question 6, determine which field (or fields) should be the primary key. Justify your choice.

10 Why are unique identifiers used in databases instead of just names?

11 What is the difference between a flat file and a relational database?

12 What is the difference between a hierarchical and a network database?

13 Explain what the term 'data integrity' means.

14 What are the three types of data anomalies? Give an example of each.

15 In relation to data storage, what is the difference between data encryption and data hashing?

16 What is the purpose of a database management system?

17 In PostgreSQL, what storage is needed for the following data types?

- a integer
- b timestamp
- c real

APPLY YOUR KNOWLEDGE

1 Using the following description of a data-driven problem, brainstorm a mind map to determine a list of functional and non-functional requirements.

Example scenario: Hervey Bay Scooter Rental

Hervey Bay Scooter Rental is a small independent operation that rents motor scooters by the hour to holiday travellers and visiting backpackers. The current system is manual in nature and requires each hirer to fill in paperwork containing their name, contact details and driver's licence number as well as who to contact in the event of an accident. A log tracks the usage of each of the business's 20 scooters. At any time, each scooter is considered to be either checked in, checked out, available or undergoing maintenance. Maintenance records and fuel usage information are scattered across files and are difficult to quantify. The owners are struggling to determine whether certain scooters are costing

more to run than they make from hiring them out. The owners would like a simple solution that stores all data in a logical, easy-to-find way that is accessible from a computer stationed on the counter at the hire office. They would like it to be menu-driven and logical in relation to their main tasks. The business has a distinctive purple and orange colour scheme and a logo incorporating an illustration of a whale. The business is small and thus has limited funds to produce a system from scratch. The owners hope to use software already contained in their computer. The computer system at the hire office is an Apple Mac desktop and has the Mac version of Microsoft Office installed.

2 To improve the accuracy of electronic records in an organisation, the human resources department wishes to create a form to acquire lists of qualifications, certifications and work history of employees. Develop an input form that could be given to each employee to harvest



TYPE	BIRD	IMAGE
Bird of prey	Owl	
Bird of prey	Eagle	
Bird of prey	Falcon	
Bird of prey	Hawk	
Perching bird	Finch	
Perching bird	Dove	
Flightless bird	Ostrich	
Flightless bird	Rhea	
Flightless bird	Cassowary	
Flightless bird	Emu	
Wading bird	Duck	
Wading bird	Swan	
Wading bird	Goose	

this data. In developing this data acquisition tool, think about how it can be best structured to be easy for people to use and to be entered logically into a database system.

- Using the sample dataset in the table of bird species (left), explore and develop ideas for a mobile user interface that will incorporate the data. Devise three possible options and highlight visual communication principles and elements in each one. Select one option and justify your choice.
- Using a programming development environment of your choice, research what modules, libraries, code references or add-on features are required to hash data. Using skills acquired in Unit 1, develop an algorithm to accept a piece of confidential data from a user and hash it. Using this algorithm, generate a coded module and a simple interface for trialling and testing. Output the hashed value back to the user to confirm its success.
- Consider the issues of security and protection relating to a school administration system that contains all information about students, teachers, subjects, subject timetables, achievement results and parents. The system is available for school administrators, teachers, parents and students to access. Government agencies also wish to access this system for statistical summarised information about the number of students, their backgrounds and the spoken languages of families. Determine some of the security and protection requirements and solutions that might be used to allow different stakeholders to utilise this system.

2.2

DEVELOPING DATA-DRIVEN APPLICATIONS AND SOLUTIONS

In this section, the methods and approaches around the development of data-driven applications and solutions will be explored, and concepts introduced in the previous section will be applied to activities and scenarios. To examine the developmental features of a solution, a systems thinking approach will be taken to identify and recognise components across human, computer, application and data systems.

As part of this systems approach, data flow diagrams will be explored and demonstrated as a means of examining systems processes and the flow of data within solutions. Information about how these diagrams can be used to develop algorithms will also be highlighted. To assist with developing relational databases, a practical guided approach will be explored, with demonstrations of approaches to exploring data needs, collecting data, developing relational tables and stepping through the normalisation process. A web context scenario will be used to demonstrate different aspects of developing data-driven components.

Data-driven systems thinking

As discussed previously, data-driven solutions are based on a number of subsystems working together to provide a viable solution to a given problem. Each of these systems can play a key part in the implementation of a successful data-driven solution.

The subsystems of a data-driven system can be many, and they may vary slightly from case to case. In a general sense, the core subsystems of a data-driven solution involve human systems, application systems and data systems.

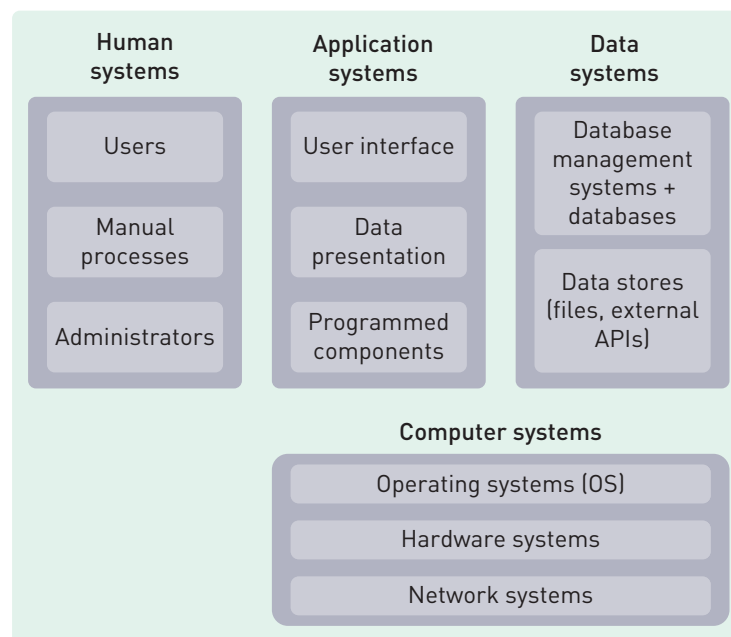


Figure 2.2.1 The general subsystems of a data-driven solution

Human systems

Human or user systems are one of the key systems that make a successful data-driven solution. But what do we mean when we refer to a 'human system'? We are referring to a user providing input or receiving output; we may also be referring to processes that occur outside the application processing or data subsystem itself. Take, for example, the creation of a data-driven solution for assigning graduating students to university courses. One input might be a student's personal details and their associated course preferences. Another input might be the student's performance rank, and this would come from the curriculum and assessment authority. Universities would input course details such as the number of places available and course requirements. The information is then collectively stored and processed. Students and university personnel can then be provided with the processed information as a user output. All of these examples in some sense involve a human or user system. As the user's needs and requirements are the fundamental reason for the existence of data-driven solutions, many human systems predominantly serve as inputs and outputs to an overall system.

i The automation or reduction of human systems is a goal within many organisations. The ability to directly exchange data between systems has emerged as one possible way to achieve this. Data needed in one system can be accessed directly from another system where data-exchange processes have been established, removing the need to rely on a human system or process to complete data flows or data dependencies. Data exchange will be a focus in Unit 4, building on your studies in Unit 2.

Application systems

Application systems act as the interface between the user and the data system. Application systems validate and process user input and translate user requests for data into a format compatible with the data system. When the data system returns a data structure, the application system presents it in a way that the user can easily understand.

Applications deal with input, processing and output. Users may be required to input data via the application. It then may process and validate this data before formatting it for action by the data system. Conversely, the application may retrieve data from a data store controlled by the data system, and process a data representation to suit a user's needs. An example of this might be that university personnel wish to see in a snapshot how many students have been allocated their first choice for the Bachelor of Engineering course at their university. The application will retrieve and process data based on the user's inputs. The data will be processed and presented as a graphical chart or summarised diagrammatically to help the user to gain insights from the data.

The examples of applications development environments across the contexts will be discussed further in later sections.

Computer systems: OS, hardware and networks

Both application and data systems are directly supported by computing systems. Computing systems are made up of specific operating systems (OS) and physical hardware systems, which may be linked together via network systems. When considering a data-driven solution, it is important to understand how application and data systems will depend on these subsystems. For example, if a web application platform such as ASP.NET was chosen with Microsoft SQL Server, Microsoft operating systems on compatible servers with network access would be recommended. Conversely, if

i There are three main implementation approaches to deploying a data-driven solution.

- 1 Localised: In this approach the database is installed on the same machine as the application and all access to the database is controlled by a single instance of the application.
- 2 Networked: In this approach the database is stored on a server accessible from multiple clients via a local network. Specialised DBMS software manages concurrent access to the database from multiple trusted instances of the application and maintains database integrity.
- 3 Cloud-based: In this approach the database is deployed on an internet accessible server (usually behind a firewall) and remote clients must establish an authenticated connection to the database before sending SQL or API formatted requests to the data server.

Localised solutions are easiest to implement but not very scalable. The best implementation approach will often depend on the technology context, the extent of data sharing required and the expected number and distribution of users.

an intelligent system solution was being planned, a certain type of microprocessor firmware and network accessories might be selected to interface with data systems stored in PCs or web services.

Data systems

The data system deals with the management and storage of data. In many cases, this will be an information system in which a database management system and a database work together. Other data stores can also be used, such as direct file storage or data exchange services. Data exchange will be explored further in Unit 4.

Data flows of data-driven solutions

One of the key considerations in planning, analysing and evaluating a potential system is to correctly document the inputs, processes and outputs of the overall system and how data is used. One method used to achieve this is to document a system in a diagrammatic form, such as with data flow diagrams. Data flow diagrams (DFDs) are primarily concerned with the process of how data flows through the system, not specifically with how data is organised within a data store. The advantage of first looking at a potential data-driven solution in this way is that the developer can easily segment different aspects of use, document data and user system dependencies and also highlight critical processes that must occur. Data flow diagrams also clearly designate linkages between subsystems and show how they must work together to generate the solution.

There are a number of graphical software tools that can assist with developing diagrams such as data flow diagrams. Possible choices include:

- Microsoft Visio
- SmartDraw
- Lucidchart
- Coggle.it
- Clickcharts
- Bubbl.us
- Creatly.com
- Loopy.

Characteristics of data flow diagrams

Data flow diagrams are made up of a number of key components. These are shown in table 2.2.1.

Table 2.2.1 Components of data flow diagrams	
D1Student data Figure 2.2.2a Data stores	The data store symbol highlights when a data store is used within a system. It can refer to a database or storage file or potentially some other data storage element. It may have a qualifying identifier (e.g. D1) to highlight that different data stores exist within a data flow diagram. The identifier designates whether the data store is digital (D), manual (M) or temporary (T).
idCommit student data Figure 2.2.2b System process	A process represents when something has to be done with the data. Processes can be decomposed into lower level DFDs but generally the process description is at a high level. DFD processes focus on movement of data through the system rather than computational logic such as that found in an algorithm. The process listed can be a marker for an algorithm to be written or documented at a later stage and nearly always begins with a verb.
Student Figure 2.2.2c External entity	An external entity refers to the sources or destination of data and is generally associated with human or other external systems beyond the scope of the current project. They detail external interactions with the system. External entities link directly to a system process.
Figure 2.2.2d Data flows	Data flows detail the connection and sequence of flows between the external entities and system processes as well as the flows between the system processes and data stores. They usually detail the 'direction' of data flow in one direction. Noun phrases may also accompany the data flow to describe the data elements within the flow.

2.2.1 Microsoft
visio
2.2.2
SmartDraw
2.2.3 Bubbl.us
2.2.4 Lucidchart
2.2.5 Creatly
2.2.6 Coggle
2.2.7 Loopy

Data flow diagrams can start as very high-level diagrams. These are sometimes referred to as context diagrams, in which the process is simply the application as a whole. Using decomposition, they can then be redrafted to lower levels (1 to 8) to contain more detailed processes and data flows. This process will be explored further in Unit 3. In Unit 2, data flow diagrams will be explored to levels 1 and 2 only.

For smaller projects, a single detailed data flow diagram can represent the entire data-driven system and its associated flows. However, in more complex projects, a number of DFDs may be required. For example, in the university course selection system mentioned earlier, DFDs may exist for big-picture system processes for the collection of student information and course selections shown in figure 2.2.3, as well as for reviewing the courses offered to students and their confirmation of acceptance (figure 2.2.4).

RULES FOR CREATING DFDs

- 1 Use no more than seven processes in a single data flow diagram.
- 2 Each process must have at least one data flow in and one data flow out.
- 3 Every data flow must connect to at least one process.
- 4 Every entity must connect to at least one process.
- 5 Data flows in only one direction.
- 6 Data flow labels describe information, not actions.
- 7 Processes are numbered according to their level.
- 8 Process names are usually verbs.
- 9 Data flows and data stores are usually nouns.

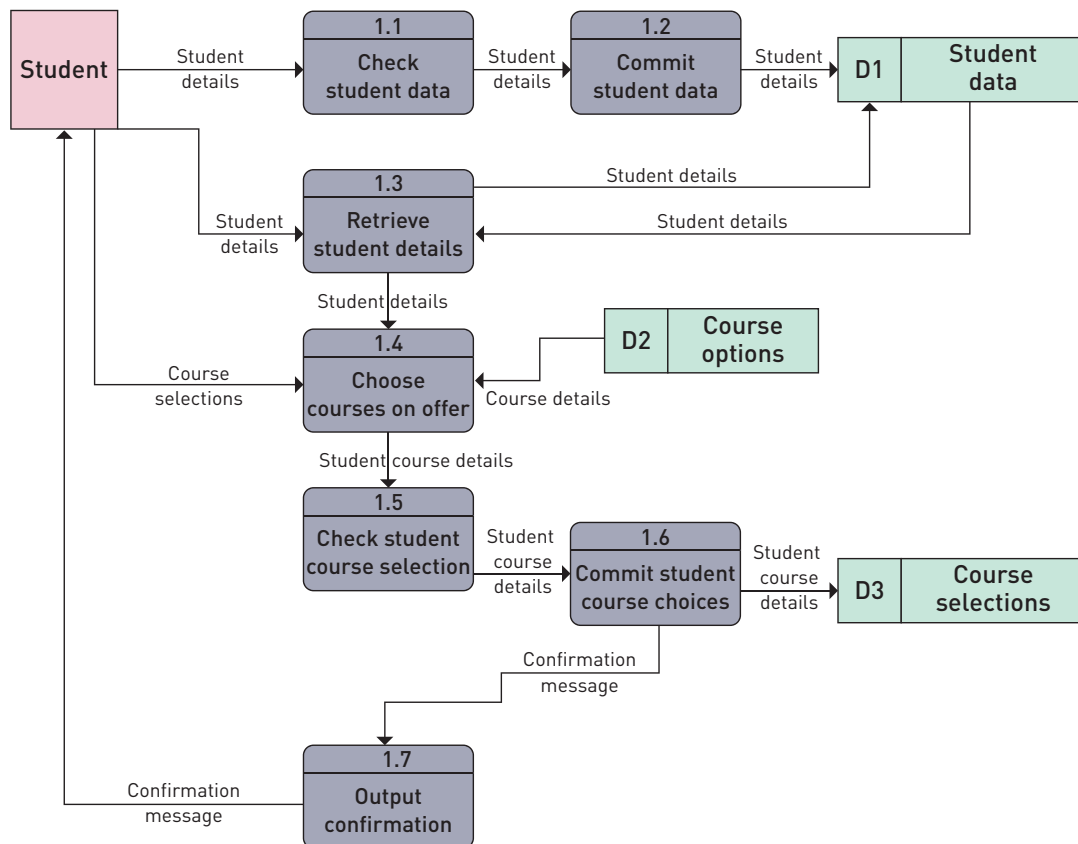


Figure 2.2.3 Data flow diagram for student information and course selections

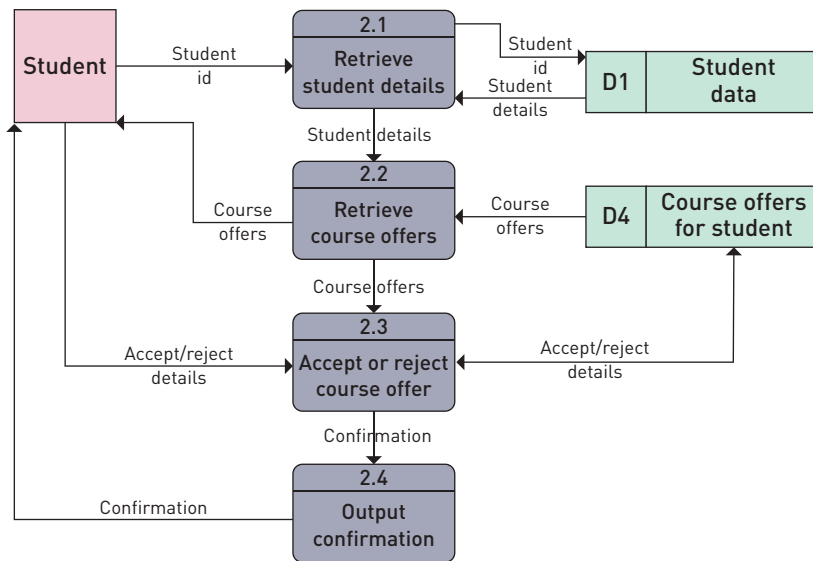


Figure 2.2.4 Data flow diagram for student course offering review and acceptance

Other processes that could be documented as DFDs include:

- university course entry and requirements
- tertiary rank calculations and entry
- course preference and acceptance processing
- competitive analysis of confirmed course selections.

Breaking these into different DFDs using computational thinking assists in decomposing the problem into smaller parts. It is essential that DFDs capture all the data flows and processes that will occur throughout the system so that the requirements of the solution can be accurately determined and planning for generation of the data and coded components of the solution can successfully take place.

ACTIVITY 2.2.1

Use the DFD in figure 2.2.5 to answer the questions that follow.

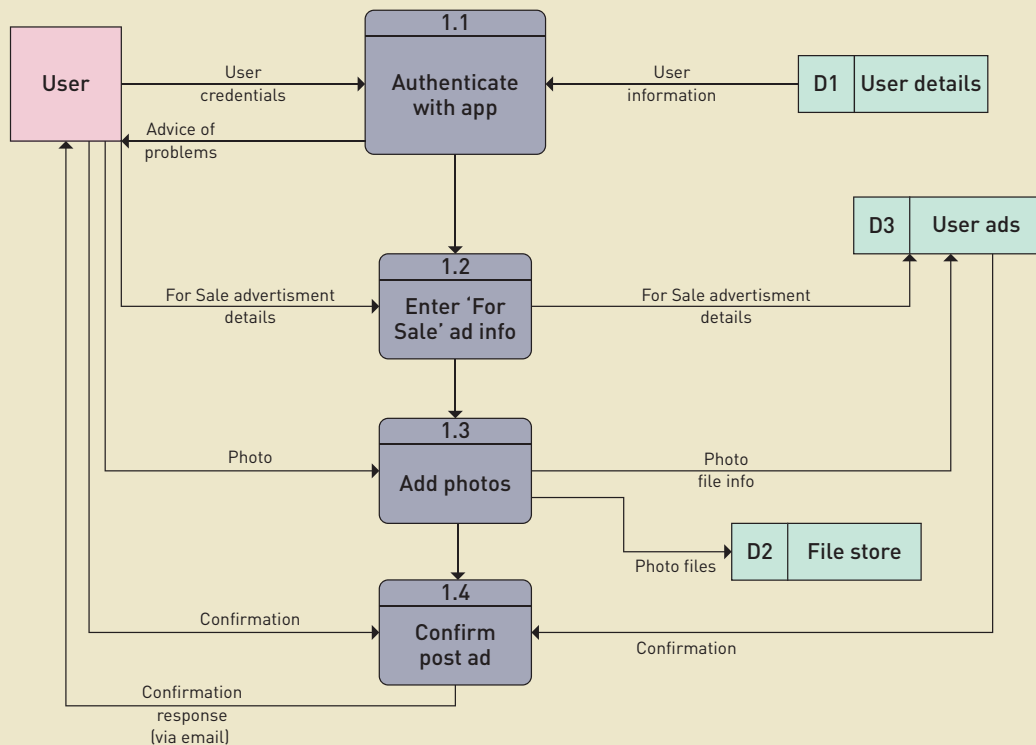


Figure 2.2.5 Data flow diagram

- 1 What or who are the inputs to this system?
- 2 What are the data stores in this data flow diagram (DFD)?
- 3 Summarise what the purpose of this data flow diagram is. (What do you think the system is about?)
- 4 Create an additional DFD that deals with a user making an offer on the advertised product.

CASE STUDY

WEB CONTEXT SCENARIO: GARDEN EXPRESS (CONTINUED FROM PAGE 93)

The Garden Express site has a number of data flows that had to be identified during the development process. The investigation of this new system involved a system designer gathering information from a number of sources to explore a list of functional and non-functional requirements (see section 2.1 relating to exploration of the Garden Express website). These sources included:

- investigation and planning notes
- client interviews
- research of existing systems for user interactions
- existing business workflows
- interface sketching and iterative planning.

In this scenario, the data flow diagrams have been grouped into different user experience workflows or tasks. The exploration of DFDs in this sample scenario is a snapshot of some of the core workflows, and more are possible. The DFDs that will be explored are:

- customers:
 - 1 creating an account
 - 2 updating an account
 - 3 making an order
- employees:
 - 4 processing an order
 - 5 updating stocked items/updating sale items.

It is worth noting that the development of the user experience design may go hand in hand with the development of data flow diagrams. The user experience design may require further adjustments due to new elements or processes being introduced.

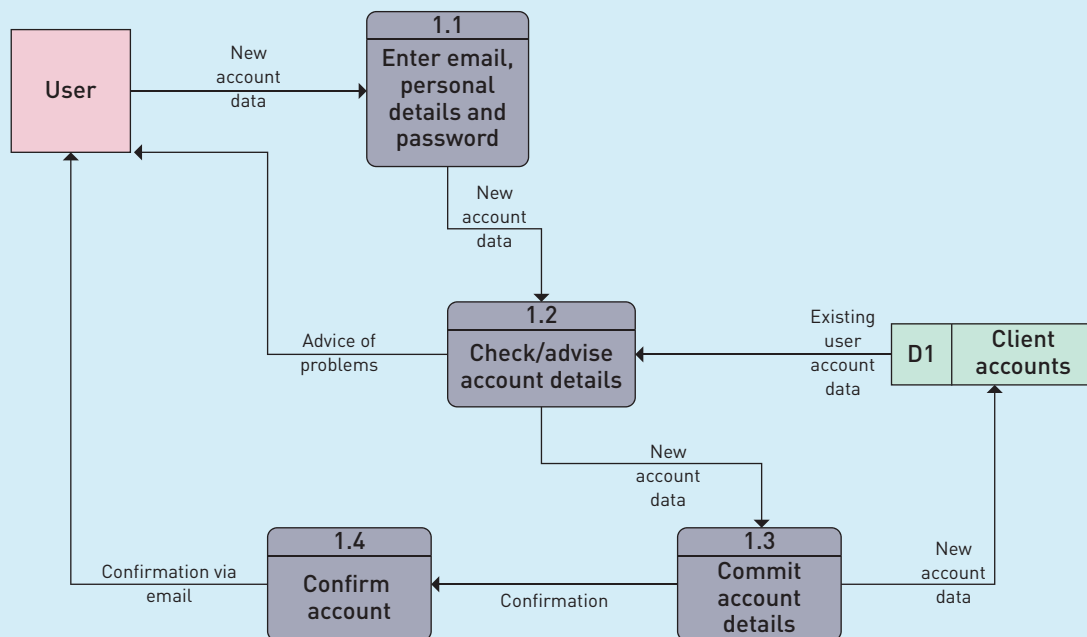


Figure 2.2.6 Data flow diagram for creating an account

In this data flow diagram, the user provides input and is either sent advisory information (something wrong with data supplied) or a confirmation of the creation of an account. The processes highlight the steps taken to complete and commit the new data to the system. Only one data store (D1) is used to check and commit new data.

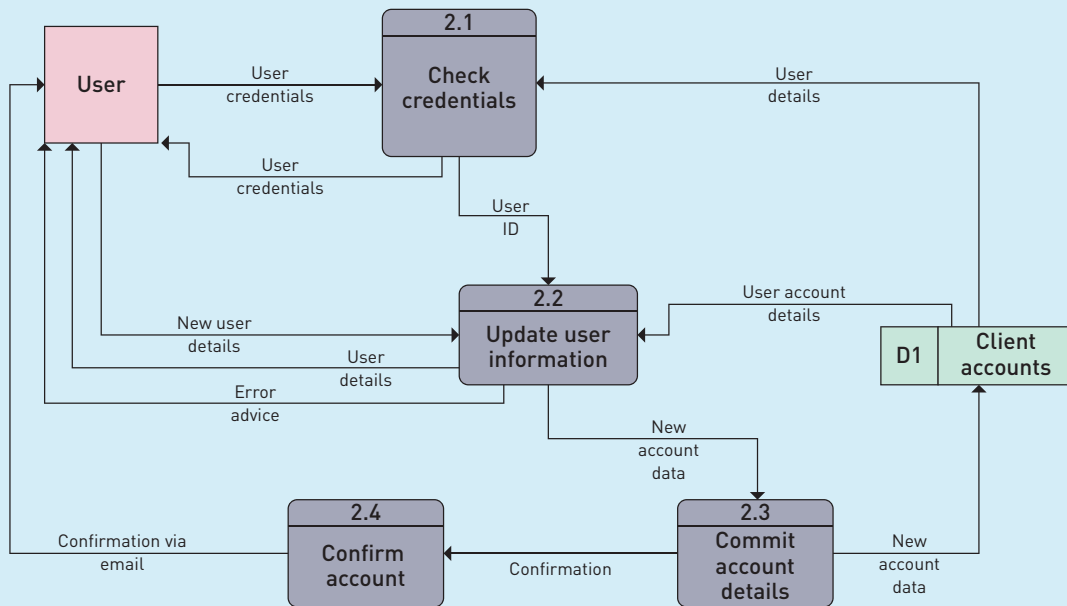


Figure 2.2.7 Data flow diagram for updating an account

In this data flow diagram, the user provides their credentials as a security and identity requirement of the system. These credentials are checked and compared against stored records before access is granted. User information is then committed to the data store (D1) and confirmation is provided back to the user.

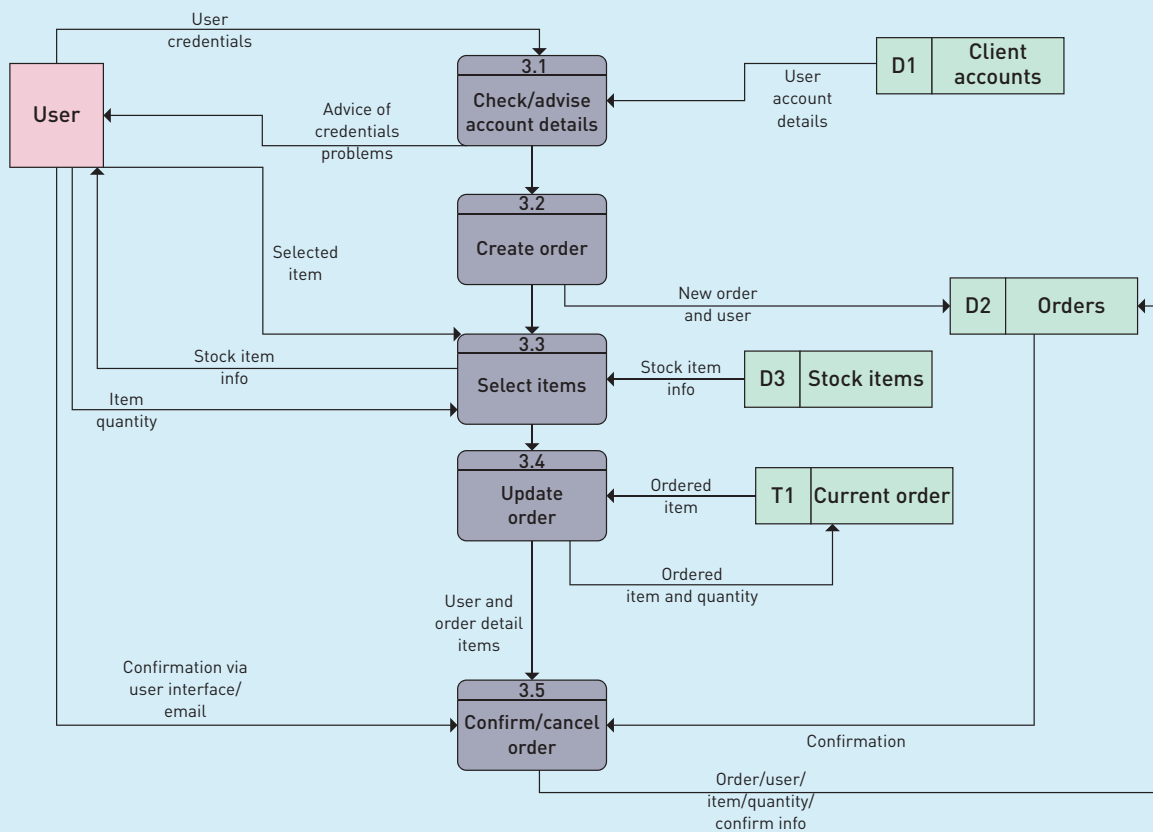


Figure 2.2.8 Data flow diagram for making an order

This data flow diagram is more complicated than most as it documents the core functionality of the ordering process. Note that between 3.3 (selecting items to order) and 3.4 (updating the order) the user will be continuing to explore the website and select items. It is envisaged that during the 3.5 process, the user will adjust options in a temporary order and ultimately confirm or cancel the order before it is stored permanently in the order database.

It is important to note that data flow diagrams are fluid in their design and can be updated at any time. It is expected that, as a project is generated, the ongoing evaluation of user experience requirements may require identified inputs, processing, storage and outputs to be refined or adjusted.

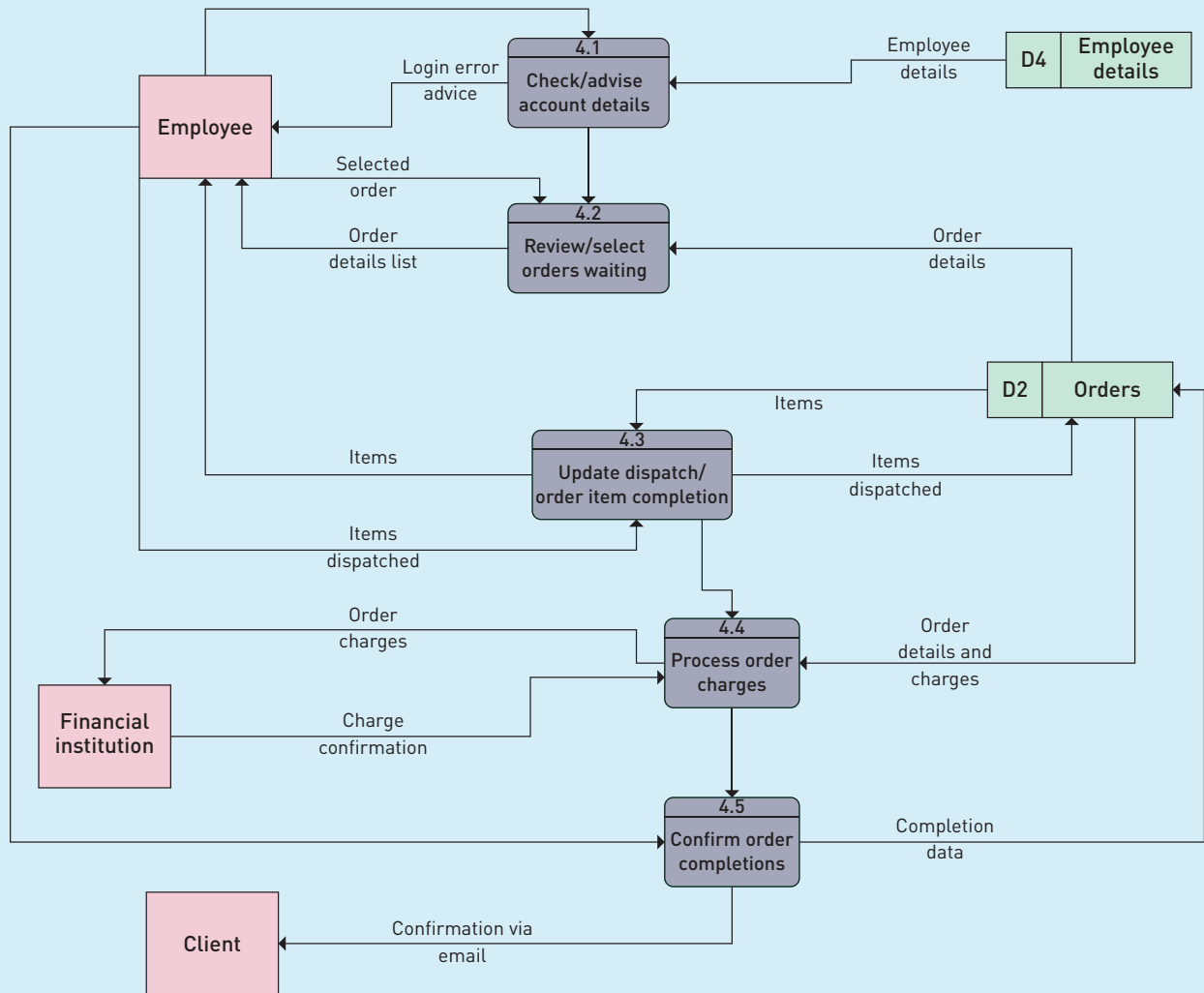


Figure 2.2.9 Data flow diagram for processing an order

This data flow diagram shows multiple external entities and how their inputs and outputs are processed to deliver the desired workflow and data flows.

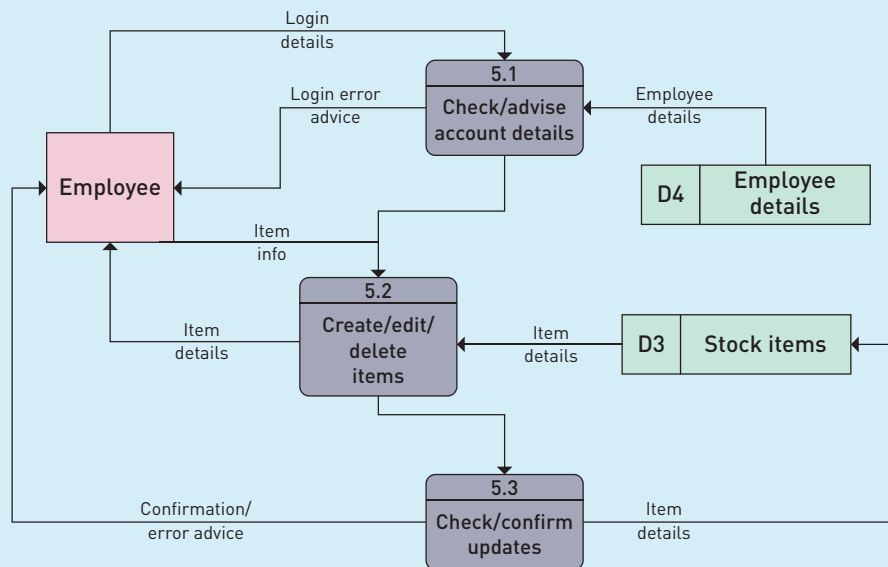


Figure 2.2.10 Data flow diagram for updating stocked items/stocked items

In this data flow diagram, after employees are validated, they can update stocked items in the system. This may include entering new items, disabling old items or updating existing ones. The updating may include selecting or unselecting items for sale.

Developing algorithms for data-driven solutions

As covered in Unit 1, an algorithm is a set of step-by-step instructions that detail a specific problem that can be translated into code. When a data-driven solution is being developed, data flow diagrams reveal processes and data flows across the entire system. As a process identifies a programmed component, an algorithm can be developed to describe the process logic.

Table 2.2.2 Algorithm for updating an account in the form of an IPO chart

Inputs	Process	Outputs
User ID User account details New user details New user details	Algorithm for process 2.2 Update account details <pre> SET DbRequest TO SELECT user details FROM client accounts data using User ID EXECUTE DbRequest storing result as User details OUTPUT User details to user INPUT New user details from user FOR EACH field of New user details IF field is valid SAVE field in New account data ELSE OUTPUT Error advice ENDIF ENDFOR OUTPUT New account data TO process 2.3 Commit Account Details </pre>	User details New account data Error advice

Processes in data flow diagrams highlight aspects of input, process and output. As computational processes deal with input, processing and output, IPO diagrams can assist with unpacking the algorithmic problem to develop solutions. An IPO diagram is simply an organiser to show the inputs coming to the process, the algorithm as the process, and the outputs that lead from the process.

Input	Process	Output
List of inputs	Algorithm	List of outputs

For specific details regarding how to construct algorithms and to use pseudocode best practices, see section 1.3.



Using the ‘update user account’ data flow diagram (figure 2.2.7), an example algorithm in the form of an IPO is shown in table 2.2.2.

Developing databases for data-driven solutions

Each data store in a data flow diagram can be implemented as flat files or tables in a database. When developing a database structure, it is important to consider the full collection of data fields that need to be maintained for the digital solution.

Once the need for a database is clear, consideration and analysis of the data requirements is needed before tables are organised and developed.

The following process is a practical guide to developing a relational database model to a given problem.

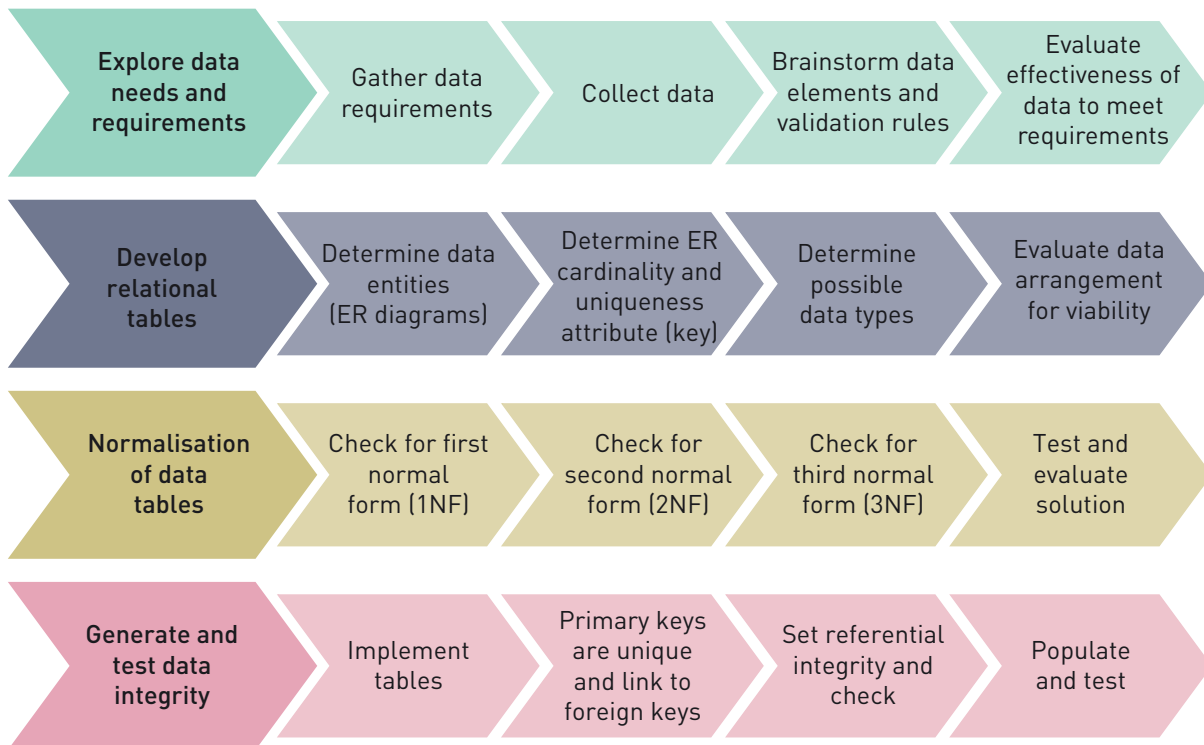


Figure 2.2.11 Analysis of data requirements

A requirement of the syllabus is that databases must be fit-for-purpose and robust, and be stepped through the normalisation process. This process is described later in this section, as is the development of relational tables including entity-relationship (ER) diagrams. While the study of these or other relational table design processes is not mandated in Unit 2, they have been included to support ad-hoc database development in the future.



The guide is not meant to be exhaustive or highly prescriptive in nature. Its goal is to support a developer's thinking about establishing a database plan for implementation within a data-driven solution. In this section, we will look at some of the phases of this process as part of planning a relational database. This will include exploring the data needs and requirements, the development of relational tables and the normalisation of data tables. The generation phase will be explored in later sections.

Explore data needs and requirements

It is important to gather as much knowledge about the data needs of a system as possible before developing a database. Much of this information should have already been collected and explored. The overall solution requirements will provide guidance for this. The data flow diagrams will also provide detailed information about what data is needed across the data-driven system.

As part of the database exploration, it may be appropriate to explore and review existing data capture tools such as forms, existing spreadsheets or logs as well as feedback from key users and project owners. This gathering process gives opportunities to determine the different required sets of like data that will form data entities at the next stage of development.

It is through this collection and brainstorming approach that individual pieces of data may be iteratively discovered and listed. To assist in the development, it may be helpful to try and group these pieces of data together.

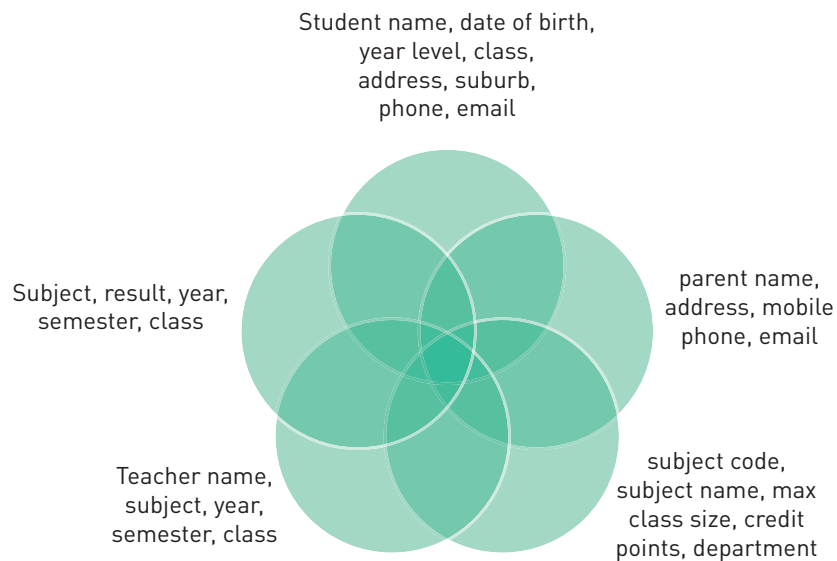


Figure 2.2.12 Exploring existing data in a school information system

It is also useful to begin thinking about the validation rules that could be applied to the data. These rules would be used to check data for validity upon entry or update. The rules may be enacted in the database management system itself, in the application, or in both. For example, a year level data element within a school database may only accept numbers between 0 and 12, a semester data element may only accept the numbers 1 and 2, email addresses must contain the '@' symbol and so on.

At this stage it is important to review the effectiveness of the data that has been selected and see that it matches the requirements of the problem. An easy and effective method to achieve this is to identify criteria from the requirements and test each element. It is good practice within assessment in Digital Solutions to log ongoing evaluative judgements when solving the digital problem.

Develop relational tables

Entity relationship diagrams

Developing entity relationship (ER) diagrams is a useful process for designing robust and viable relational databases. Database designers use ER diagrams to diagrammatically map details of connections between major key data (entities) and their associated attributes (contributing data). They assist with describing different data and the relationships between these elements that will ultimately be transferred to a real-life database. Entity relationship diagrams have become widely used since the model was first created in the mid-1970s by Peter Chen.

Different styles of ER diagrams now exist from the initial version and there are a number of ways to represent them. The style that is represented in this book is a classic extended Chen style where attributes are represented as ovals and attached to the relevant entity or relationship. Other variations such as the Information Engineering model are also acceptable practice. This other model documents attributes inside the polygon-shaped entity model. When researching, you may notice that some software packages represent this differently, but in a general sense the diamond shapes showing relationship and direct lines identifying cardinality between the entities still exist.

While it is recommended that you sketch out rough versions of your entity relationship diagrams to assist in drafting, there are several applications that can be used. These can be easily found via a web search. Below is a list of applications that may be useful. Some provide some form of free access; others require provision of licensing.

- Clickcharts for Windows
- Clickcharts for Mac OS
- Lucidchart online web tool.



2.2.8
Developing
entity
relationship
diagrams
2.2.9 NCH
software
2.2.10
Lucidchart

Entities

In extended Chen model style, entities are presented as rectangles (right).

An entity can best be thought of as a definable thing or concept that will exist within the context of the database. It is best to think of entities as nouns. Ultimately, entities should be your tables in the relational database, so it is sometimes useful to frame entities as contextualised future tables. Examples of entities could be a person or type of person (e.g. a student), an object (e.g. an order), a concept from combined data (e.g. a student profile) or a definable event (e.g. a semester).

ENTITY

Attributes

Attributes that are associated with entities or relationships are represented as ovals (right).

An attribute is a property or a characteristic of the entity that can hold an element of data. In a database, these would end up being the fields/columns in a table. While the information engineering model may also include elements such as data types for attributes, the extended Chen model simply lists the attribute names.

Attribute

Relationships

Relationships in this model are represented as diamonds (right).

Relationship

One-to-one:	1 _____ 1	or	+ ——— +
One-to-many:	1 _____ M	or	+ ——— <
Many-to-many:	M _____ M	or	> ——— <

Figure 2.2.13 Representing cardinality lines in ER diagrams

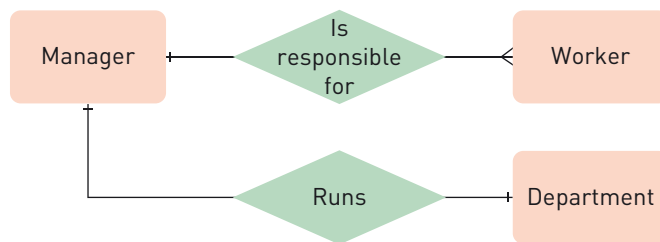


Figure 2.2.14 ER diagram for a manager and their department



Figure 2.2.15 ER diagram (one-to-many)



Figure 2.2.16 ER diagram (one-to-one)

range from nothing (0) up to one. So for every *one* manager, there could be *up to one* department, and vice versa. We define this as a one-to-one cardinality relationship (see figure 2.2.16).

We will explore this approach to developing relational tables within the Garden Express case study.

Cardinality lines that connect the relational shapes between the entities can each be represented in two ways, as shown in figure 2.2.13.

A relationship is used between two entities to show how they are associated with each other. For example, a workplace manager may be responsible for some workers in their department. When we put the three ER elements together to signify this, we may get the arrangement shown in figure 2.2.14.

Cardinality is a fundamental concept when dealing with relational databases. Its purpose is to define the number of occurrences of one entity that can be enacted for a single occurrence of the related entity. In the previous example, a manager may have many workers, or they may not have any, depending on the nature of their role.

The works manager may have a team of people, but the HR manager might work alone. The cardinality relationship between the manager and the workers could range from nothing (0) to any possible number (many). In other words, for every *one* manager, there could be *many* workers. We define this as a one-to-many cardinality relationship (see figure 2.2.15).

The requirement of this system is that there is only up to one manager for a department, and only up to one department for a manager. This means that the cardinality relationship between the manager and the department could

DEVELOPING AN ER DIAGRAM

A simple approach to developing an ER diagram is to think about the relationships between entities and then think about the context of how examples may relate.

- 1 Identify the 'big' entities.
- 2 Relate them using relationship 'phrases'.
- 3 Use examples to discover the cardinality of the relationship.
- 4 Consolidate many-to-many relationships.
- 5 Add in extra data via attributes.

CASE STUDY

WEB CONTEXT SCENARIO: GARDEN EXPRESS (CONTINUED FROM PAGE 130)

The Garden Express site depends on a custom-designed database to dynamically construct web pages (content is stored in the database and served via script-enabled web pages) and to track users, orders and products. To this point, a detailed exploration of the problem has occurred along with the identification of functional and non-functional requirements. The user experience has been investigated and sketches of the solution have been developed. Data flows have also been tracked and documented in the form of data flow diagrams. An investigation of data requirements has been completed, identifying key data elements.

Data needs and requirements

As part of the analysis of this data-driven problem, the following data requirements have been gathered from the available documents, interviews, identified needs and from prior documentation.

ACTIVITY 2.2.2

Organise the following data elements identified for the Garden Express database into similar groupings: customer, products, categories, suppliers, orders, customer username, customer email, customer password, customer first name, customer last name, customer address, customer suburb, customer telephone, product code, product name, product description, product information, product retail price, product cost, product sale price, product category code, product category description, product category grouping, product supplier, supplier contact name, supplier address, supplier suburb, supplier postcode, customer order number, product ordered, number of products on order, price on order, date of sale, delivery address, delivery suburb, delivery post code, total due, payment type, payment amount, payment reference, courier code, courier name, courier cost, courier tracking code, employee username, employee name, employee password, employee email.

Develop relational tables

The first step to developing tables is to map the data through an ER model. We will follow the suggested steps described above.

- 1 Identify the 'big' entities.

From these attributes of data, we can identify some entities for starting our entity relationship diagram (see figure 2.2.17).

Customers

Orders

Products

Categories

Employees

Figure 2.2.17 Entities for an ER diagram

2 Relate them using relationship 'phrases'.

We think about the way in which these entities relate directly to each other. It is important to only identify direct relationships that are necessary.

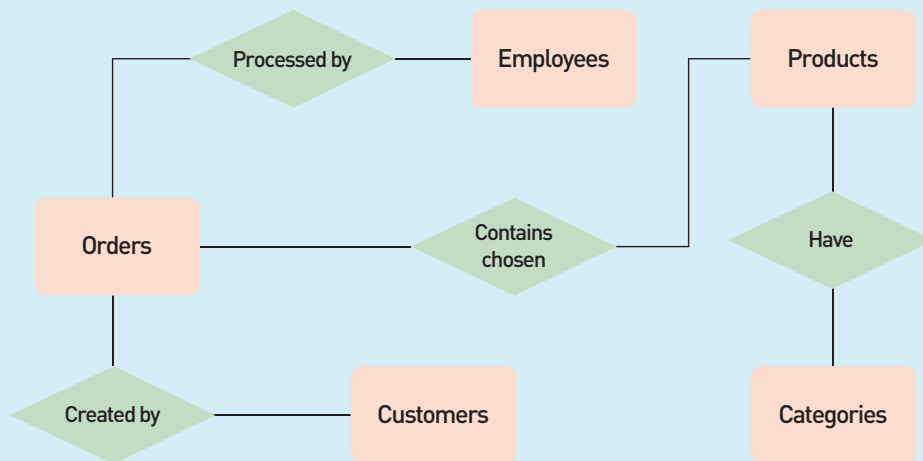


Figure 2.2.18 Relationships between the entities

3 Use examples to discover the cardinality of the relationship.

We think about how the cardinality should be applied. We ask questions and think of data examples.

- Can customers have multiple orders?
- Do orders only have one customer?
- Can an order only be updated by one employee?
- Can an order contain multiple products?
- Can products be in multiple orders?
- Do products have only one category?
- Can a category have multiple products?

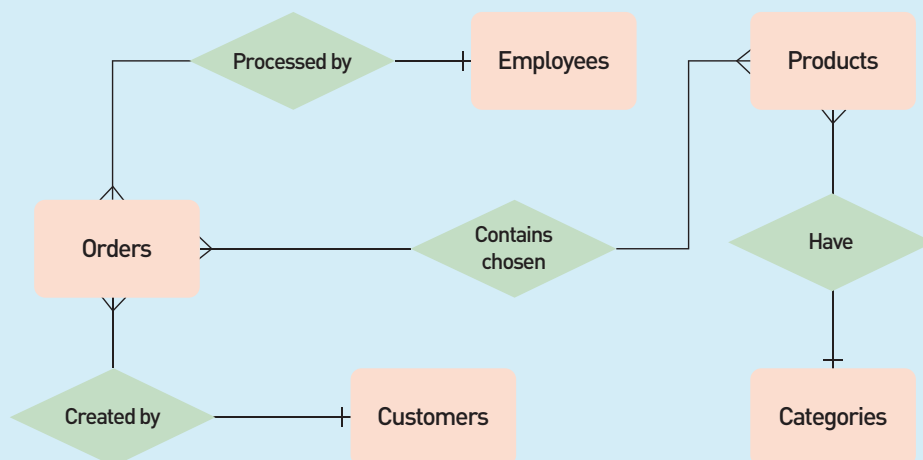


Figure 2.2.19 ER diagram with cardinality applied

4 Consolidate many-to-many relationships.

As alluded to, many-to-many relationships are not something that can be left unresolved in an entity relationship model. A many-to-many relationship indicates that there is a need for a composite entity that can have attributes that require both entities to make sense.

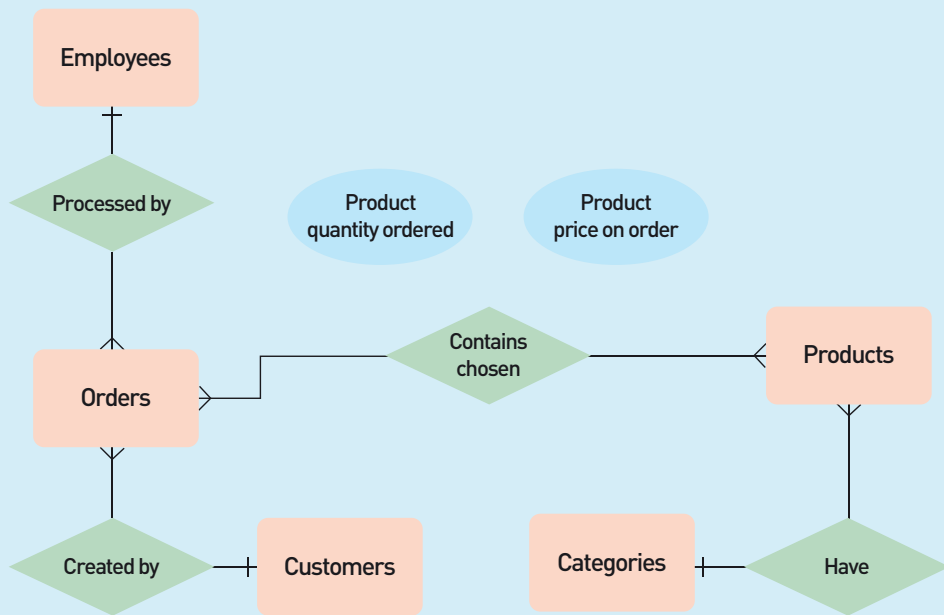


Figure 2.2.20 Consolidating many-to-many relationships

In figure 2.2.20, you will notice that two attributes have been added prematurely to show the problem faced. The 'Product quantity ordered' and the 'Product price on order' attributes cannot relate purely to the **Orders** entity because there can be multiple products. Conversely, they cannot just relate to the **Products** entity as it is more concerned with what the products are. These attributes need both. Therefore, a new composite entity must be created.

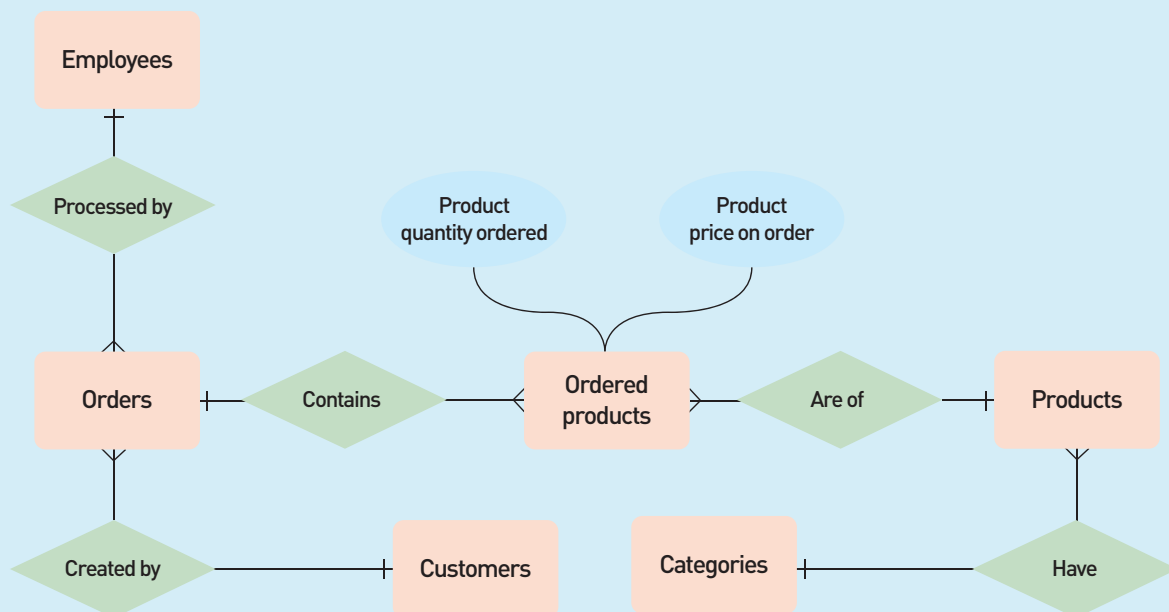


Figure 2.2.21 ER diagram with composite entity created

The reason it is called a composite entity will become obvious when assigning keys during the normalisation process. The entity (table) is destined to have two keys to make each record unique in a relational database.

5 Add in extra data via attributes.

Now that steps 1 to 4 have been successfully completed, the attributes can then be added from the list gathered.

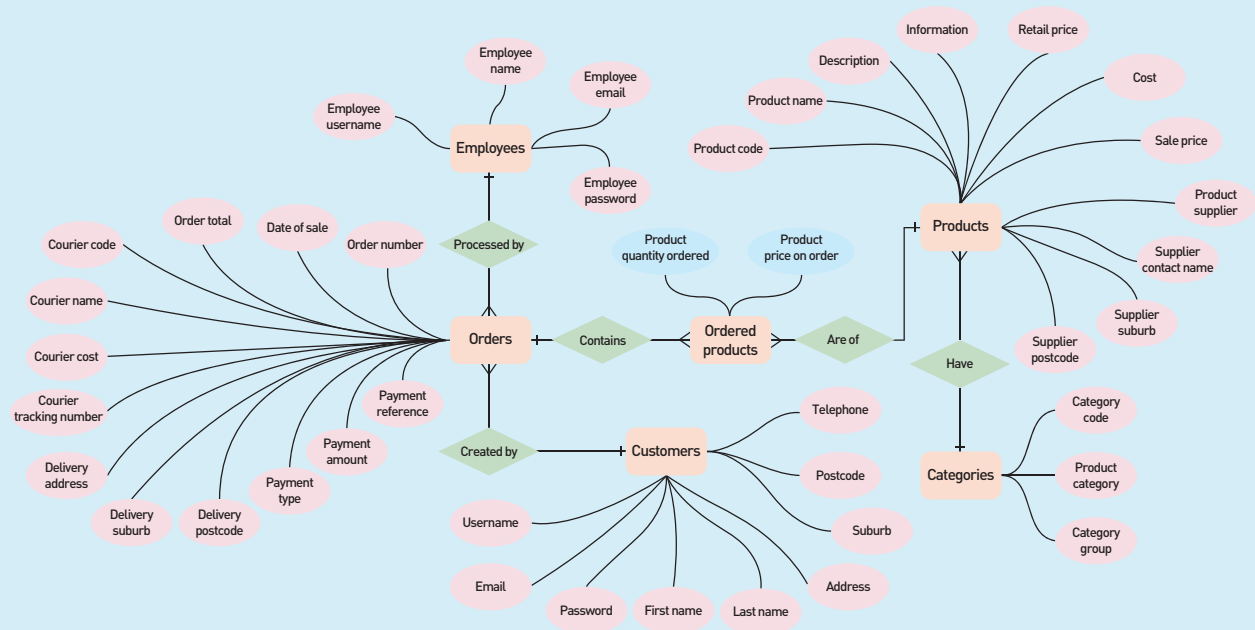


Figure 2.2.22 ER diagram with attributes added

This process may be repeated to refine, add or adjust the entity relationship diagram to reflect changes in data requirements. Further issues such as supplier details and record-keeping for processing and shipping are additional adjustments that could conceivably be made.

It is recommended, towards the end of this process, that thought be given to assigning data types for future reference in the form of a data dictionary. A data dictionary is designed to list each data item and its associated data type as well as appropriate validation rules that will be used to ensure only correct data is allowable. A validation rule might be a range, a key, or simply some value options. Example data is also supplied in some cases.

While specific data types may need to be adjusted to suit the chosen database management system, globally accepted generic data types (or data type descriptions) can be used initially. Examples of data dictionaries associated with Garden Express customers and order details attributes are listed in tables 2.2.3 and 2.2.4.

Table 2.2.3 Customer table

FIELD	DATA TYPE	VALIDATION RULES	EXAMPLE DATA
Username	NVarChar (20)	Primary key	123
Email	NVarChar (150)	Contains '@'	sdc@dsc.com
Password	NVarChar (35)		1f8094bb6297edadc7755127ec8c8d1d
first_name	NVarChar (100)		David
last_name	NVarChar (100)		Jones
address	NVarChar (200)		65 Department Store Way
suburb	NVarChar (50)		Sydney
post_code	NVarChar (10)	9999	2001
telephone	NVarChar (25)	99-9999-9999	02-9983-3254

Table 2.2.4 Order details table

FIELD	DATA TYPE	VALIDATION RULES	EXAMPLE DATA
order_number	Integer	Primary key	1
date_of_sale	Date	YYYY-MM-DD	2018-04-02
order_total	Float		23.45
courier_code	NVarChar (10)	Select from list	ABC
courier_name	NVarChar (100)		ABC Couriers
courier_cost	Float		12.40
courier_track	NVarChar (50)		ABC12345A4
del_address	NVarChar (200)		26 John Street
del_suburb	NVarChar (50)		Rockhampton
del_postcode	NVarChar (10)		4700
payment_type	NVarChar (10)		CC
payment_amount	Float		35.85
payment_ref	NVarChar (50)		1223444343sd3

At the completion of this, the entity relationship diagram and associated documentation can be used to construct tables that can be stepped through the normalisation process in readiness for generation within a chosen database management system.

Normalisation process

Having stepped through developing an entity relationship diagram for your custom collection of data, the normalisation process will be relatively simple and swift. When the ER diagram is converted to table-like structures, it is essential that the collections are checked through a normalisation process and that it meets these forms. The purpose of this process is to make your relational database as efficiently structured as possible and remove redundancy issues that can lead to data integrity problems.

There are a total of six normal forms listed as ‘first normal form’ (or 1NF for short) through to ‘sixth normal form’ (or 6NF). In Digital Solutions, students will only utilise the first three steps of the normalisation process (see table 2.2.5).

Table 2.2.5 Steps 1–3 of the normalisation process

FORM	STEP
First normal form	Each record (row of data) within a table is atomic in nature (i.e. it cannot be broken into smaller parts) and is separated into records that are different from others.
Second normal form	Each record (row of data) within a table has met the requirements of being at first normal form (1NF) and each attribute (field) is fully dependent on the primary key(s) within that record to give overall meaning.
Third normal form	Each record (row of data) within a table has met the requirements of first normal form (1NF) and second normal form (2NF) and must not have any other data dependencies other than the primary keys.

Using examples, let’s now explore the normalisation process.

First normal form (1NF)

In this example, we have been presented with sample data that does not meet first normal form. It contains duplicated values within fields and has summary data that has not been applied to all records.

Table 2.2.6 Students table that does not meet first normal form

STUDENT ID	FIRST NAME	LAST NAME	YEAR	SEMESTER	SUBJECT	RESULT	RESULT DESCRIPTION
1	Roy	Andrews	2018	1	English	A	Achieves at a Very High Standard
				2	English	B	Achieves at a High Standard
				1, 2	Math B	A	
2	Jared Lee	Baumanis	2016	1	English	B	
				1, 2	Math A	C	Achieves at the Standard
				2	Science	A	
3	Jared Hugh	Baumanis	2017	1	English	B	
				2	English	A	

To be compliant with first normal form, the table must be atomic in nature – meaning there should not be more than one value per field, or there should not be more than one column with the same kind of value. To achieve this, we need to replicate values out into fully filled records.

Table 2.2.7 Students table with one value per field

STUDENT ID (PK)	FIRST NAME	LAST NAME	YEAR (PK)	SEMESTER (PK)	SUBJECT (PK)	RESULT	RESULT DESCRIPTION
1	Roy	Andrews	2018	1	English	A	Achieves at a Very High Standard
1	Roy	Andrews	2018	2	English	B	Achieves at a High Standard
1	Roy	Andrews	2018	1	Math B	A	Achieves at a Very High Standard
1	Roy	Andrews	2018	2	Math B	A	Achieves at a Very High Standard
2	Jared Lee	Baumanis	2016	1	English	B	Achieves at a High Standard
2	Jared Lee	Baumanis	2016	1	Math A	C	Achieves at the Standard
2	Jared Lee	Baumanis	2016	2	Math A	C	Achieves at the Standard
2	Jared Lee	Baumanis	2016	2	Science	A	Achieves at a Very High Standard
3	Jared Hugh	Baumanis	2017	1	English	B	Achieves at a High Standard
3	Jared Hugh	Baumanis	2017	2	English	A	Achieves at a Very High Standard

The table is now considered to meet the requirements of first normal form.

Before proceeding to the next stage, we must now identify the field (or fields) that will be the primary key(s). The role of a primary key (or composite of keys) in a table is to give each record its uniqueness so that records can be differentiated. Primary keys can also assist with relations between tables at a later point through the use of foreign keys.

In this example, not one singular field can act as the primary key to give each record its uniqueness.

If we look at table 2.2.7, it requires four fields to make it unique: Student ID, Year, Semester and Subject. The 'Result' field is dependent on all four fields to give it true meaning as an element of data (for example, the 'A' in the top row is for Student 1 in Year 2018 during Semester 1 for English).

Second normal form (2NF)

To be compliant to second normal form, data in a table must be fully dependent on the primary key(s) with each record to give overall meaning. In other words, every field that is not a key must be dependent on the fields that are keys to give it meaning. If we look at the previous adjusted example in table 2.2.7, it is easy to see that 'Result' is fully dependent on all four fields that we have identified as keys (Student ID, Year, Semester and Subject). Similarly, 'Result description' at this stage can be dependent on all fields, but we can see it might become a problem in the future. This issue will be dealt with when reviewing third normal form. For now, the data is compliant at second normal form.

Let's now turn our attention to the other remaining non-key fields – 'First name' and 'Last name'. These are not fully dependent on all four keys. In fact, they are only dependent on one – the Student ID field. What this means is that, in its current form, this table is not compliant with second normal form and some action must now be taken.

To achieve this, we must now take out the fields that are not dependent on all fields and put them in a new, separate table (see table 2.2.8).

Table 2.2.8 Separated tables for student information and student results

A STUDENT INFORMATION TABLE					
STUDENT ID (PK)	FIRST NAME	LAST NAME			
1	Roy	Andrews			
2	Jared Lee	Baumanis			
3	Jared Hugh	Baumanis			

B STUDENT RESULTS TABLE					
STUDENT ID (PK)	YEAR (PK)	SEMESTER (PK)	SUBJECT (PK)	RESULT	RESULT DESCRIPTION
1	2018	1	English	A	Achieves at a Very High Standard
1	2018	2	English	B	Achieves at a High Standard
1	2018	1	Math B	A	Achieves at a Very High Standard
1	2018	2	Math B	A	Achieves at a Very High Standard
2	2016	1	English	B	Achieves at a High Standard
2	2016	1	Math A	C	Achieves at the Standard
2	2016	2	Math A	C	Achieves at the Standard
2	2016	2	Science	A	Achieves at a Very High Standard
3	2017	1	English	B	Achieves at a High Standard
3	2017	2	English	A	Achieves at a Very High Standard

These separated tables must now be rechecked to see that they meet both first normal form and second normal form. As they both now meet the requirements of first and second normal form, we can now move to the next stage.

Third normal form (3NF)

To be compliant with third normal form, the data within the tables must now be checked to confirm that all non-key field data is not only dependent on the associated keys, but does not have any other dependencies outside of this. This final stage, when completed correctly, will greatly assist in making your relational tables have as little redundancy as possible.

When we look at the data in table 2.2.8A, the new student information easily meets the requirements of third normal form. The 'First name' and 'Last name' fields are solely dependent on the 'Student ID' field, and not on each other, to give a combined record meaning. This is even more clearly shown because there are two students with very similar names and the importance of the Student ID to distinguish them in the system is paramount.

In the student results table (table 2.2.8B), there is a problem. It has been found to be acceptable for the 'Result description' field to be dependent on the listed primary keys, but in this situation we can also see that it has a definitive and dependent link on the 'Result' field. In other words, the data that appears in the 'Result' field dictates the data that should be in the 'Result description' field (for example, a result 'A' will be aligned to a description 'Achieves at a Very High Standard').

To achieve compliance with third normal form, action must be taken. These fields must be somehow separated into a new table and linked back through keys. The method for the separation will look similar to the separation that took place in the last stage. The 'Results' field will remain in the student information table and will link as a foreign key to a new table with 'Results' listed as the new primary key (which was identified when 1NF, 2NF and 3NF checks were reconfirmed).

Table 2.2.9 Separated tables for student information, student results and result translations

A STUDENT INFORMATION TABLE		
STUDENT ID (PK)	FIRST NAME	LAST NAME
1	Roy	Andrews
2	Jared Lee	Baumanis
3	Jared Hugh	Baumanis

C RESULT TRANSLATIONS TABLE	
RESULT (PK)	RESULT DESCRIPTION
A	Achieves at a Very High Standard
B	Achieves at a High Standard
C	Achieves at the Standard

B STUDENT RESULTS TABLE				
STUDENT ID (PK)	YEAR (PK)	SEMESTER (PK)	SUBJECT (PK)	RESULT
1	2018	1	English	A
1	2018	2	English	B
1	2018	1	Math B	A
1	2018	2	Math B	A
2	2016	1	English	B
2	2016	1	Math A	C
2	2016	2	Math A	C
2	2016	2	Science	A
3	2017	1	English	B
3	2017	2	English	A

Note that in each scenario in which a data item was separated into two tables, the number of stored data values kept was reduced. There is only one instance where an 'A' result description is stored in the result translations table; and similarly, in the student information table there is only one instance of where the first name and last name of Student 1 is stored. This provides maximum redundancy and significantly minimises the potential issues associated with data anomalies.

Evaluate the developed database plan

It is important at this point to look across your completed relational database plan and review it against the initial requirements of the solution. Care should be taken to check that it meets these requirements and whether any new success criteria should be applied to the project for future checks and reference.

It is also important to use sample data to test for data integrity issues. This may include checking that data anomalies such as insert, update or deletion anomalies can't occur. Looking for examples of redundant data (the same pieces of data entered twice in the database – other than keys) is one possible methodology to use.

If this review is a success, you are now ready to embark on generating your solutions.

CASE STUDY

WEB CONTEXT SCENARIO: GARDEN EXPRESS (CONTINUED FROM PAGE 139)

The entity relationship diagram has been used to develop tables that can now be checked for compliance with first, second and third normal forms. These tables should also include a linking field between the relationships identified. For example, in the Products table, a category foreign key will be needed to link to the Categories table. For this scenario review, only the Orders and Products tables will be examined. In these cases, some sample data has been added to simulate potential use.

ORDER_NUMBER	DATE_OF_SALE	ORDER_TOTAL	COURIER_CODE	COURIER_NAME	COURIER_COST	COURIER_TRACKING_NUMBER	DEL_ADDRESS	DEL_SUBURB	DEL_STATE	DEL_POSTCODE	PAYMENT_TYPE	PAYMENT_AMOUNT	PAYMENT_REFERENCE
1	05/01/2018	123.40	AP	Australia Post	42.50	1122X	22 Horse St	Pialba	QLD	4655	CC	165.90	123456781223
2	12/01/2018	133.80	AP	Australia Post	22.80	1133X	23 Cattle Rd	Gympie	QLD	4570	CC	156.60	123456781245
3	14/01/2018	22.45	CP	Couriers Please	8.50	XD232	24 Sheep Lane	Kepnock	QLD	4670	PP	30.95	80\$39##39DFE#\$A2

PRODUCT_CODE	PRODUCT_NAME	DESCRIPTION	INFORMATION	RETAIL_PRICE	COST	SALE_PRICE	PRODUCT_SUPPLIER	SUPPLIER_CONTACT	SUPPLIER_ADDRESS	SUPPLIER_SUBURB	SUPPLIER_STATE	SUPPLIER_POSTCODE	CATEGORY
4	Certified Seed Potato – Pink Fir Apple 1kg	Certified Seed Potatoes. This is a Pink Fir Apple variety.	Comes in a 1 kg pack and can produce up to 10 kg of crop	11.00	2.40	--	John's Seed Stock	John Smith	22 Growing Place	Wondunna	QLD	4655	Legumes
5	Seed – Chill Orange Wonder	Seed stock – Chill Orange Wonder. Summer Bloom.	100 seeds per pack. Best planted in July/ August for Summer bloom.	5.00	0.88	--	John's Seed Stock	John Smith	22 Growing Place	Wondunna	QLD	4655	Seed Stock



First normal form (1NF) compliancy test

In this case, all of the data within the two sample tables has been verified to be atomic across records. All attribute values are singular and separated as required. The initial primary keys have now also been identified. In the Orders table this has been identified as the 'Order number' and in the Products table this has been identified as the 'Product code'.

Second normal form (2NF) compliancy test

As previously explored, to meet second normal form compliance, all non-key fields must be able to fully depend on the nominated key fields. In the case of the Orders table, all non-key fields conform to this requirement. In the Products table most non-key fields are obviously compliant. Information relating to the supplier (Product_supplier, Supplier_contact, Supplier_address, Supplier_suburb, Supplier_state, Supplier_postcode) meet second normal form compliancy but are problematic for the next stage.

Third normal form (3NF) compliancy test

In this stage, all non-key fields must meet 2NF and not have any dependence on any other field. In the Orders table there are no significant issues of compliance. The Payment information is acceptable as long as the functional requirement of one single transaction per order is maintained. In relation to the Products table, the supplier information is of concern. The contacts and address details are dependent on the supplier name and therefore action must be taken to remedy this situation.

To resolve this, the supplier data must be separated into a new table and linked back to the Products table.

PRODUCT_CODE (PK)	PRODUCT_NAME	DESCRIPTION	INFORMATION	RETAIL_PRICE	COST	SALE_PRICE	SUPPLIER_CODE	CATEGORY
4	Certified Seed Potato – Pink Fir Apple 1kg	Certified Seed Potatoes. This is a Pink Fir Apple variety.	Comes in a 1 kg pack and can produce up to 10 kg of crop	11.00	2.40	--	1	Legumes
5	Seed – Chill Orange Wonder	Seed stock – Chill Orange Wonder. Summer Bloom.	100 seeds per pack. Best planted in July/August for Summer bloom.	5.00	0.88	--	1	Seed Stock

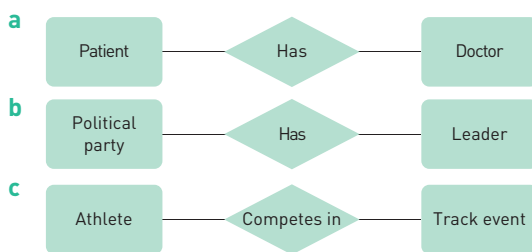
SUPPLIER_CODE	PRODUCT_SUPPLIER	SUPPLIER_CONTACT	SUPPLIER_ADDRESS	SUPPLIER_SUBURB	SUPPLIER_STATE	SUPPLIER_POSTCODE
1	John's Seed Stock	John Smith	22 Growing Place	Wondunna	QLD	4655

The supplier information has been separated into an additional table. For completeness and future uniqueness requirements, a supplier code primary/foreign key has been created. This option should also be explored for linking categories to the category table. This is because alphanumeric keys are more reliable for uniqueness throughout the life of the database.

Once all table normalisation compliance tests have been met, we are able to document our relational database table design in the form of a relational schema. At this stage, other elements such as primary/foreign keys can be confirmed as well as optional fields (allowances for null values in a record). This will be explored in the next section.

TEST YOUR KNOWLEDGE

- 1 What different types of systems are used for data-driven solutions?
- 2 What components make up a data system such as one involving relational data?
- 3 How do applications systems use data?
- 4 What is the purpose of a data flow diagram?
- 5 What are the main components of a data flow diagram?
- 6 In a data flow diagram, an external entity can have data flows directly to or from a data store – true or false?
- 7 What are the rules for creating a data flow diagram?
- 8 How does an IPO diagram work in conjunction with a data flow diagram?
- 9 What does an ER diagram show?
- 10 What are the main components of an ER diagram?
- 11 What does the term cardinality refer to?
- 12 What is the cardinality of the following ER relationships?



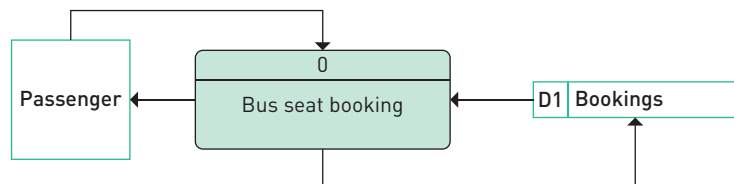
- 13 Identify the most viable generic data types for this partially completed data dictionary.

LIBRARY BOOK TABLE			
FIELD	DATA TYPE	VALIDATION RULES	EXAMPLE DATA
AssetID	NVarChar (20)		AQ0000000000000000000034323
Title			Wind in the Willows
Location			Junior Fiction
Author1			Smith, L
Author2			
ISBN			978-01700364335
Publisher			Cengage Learning Australia Pty Ltd
Publication Year			
PrintEdition			4th Edition

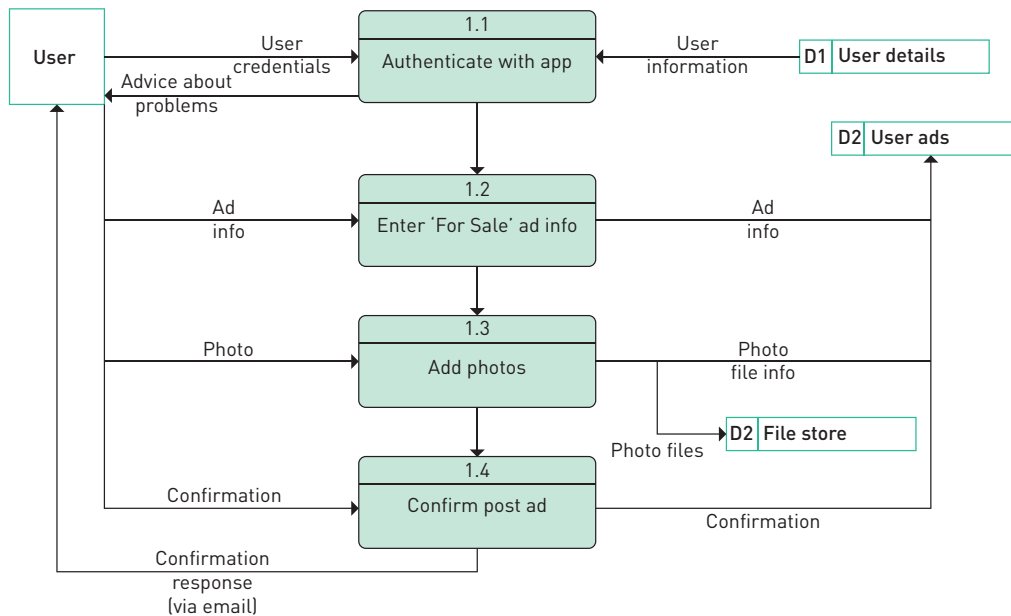
- 14 In the normalisation process of data, what does it mean to be compliant with second normal form?
- 15 In the normalisation process of data, what does it mean to be compliant with third normal form?

APPLY YOUR KNOWLEDGE

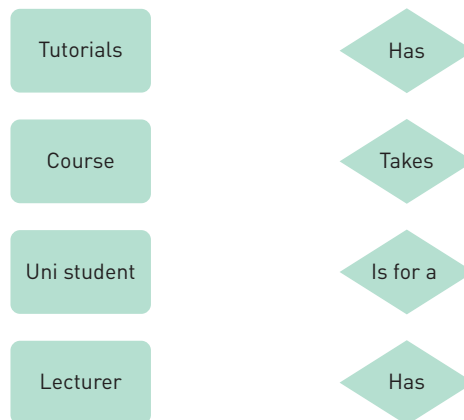
- 1 The following high-level context data flow diagram relates to a passenger booking a seat on a bus (e.g. Murray's bus scheduled between Brisbane and Toowoomba). Decompose the processes of the DFD to illustrate more detailed processes and data flows.



- 2 Using the following DFD diagram, complete an IPO chart, including a detailed algorithm of process 1.2.



- 3 Arrange the entities and relationships of the following ER diagram.



- 4 Using this sample of un-normalised data, adapt it to be compliant to first normal form.

Super Hockey Supplies

INVOICE

Invoice no: 0123
 Customer No: 2613
 Customer Name: _____
 Customer Address: _____
 Date: _____
 Order No: _____



Product No.	Product Desc.	Quantity	Unit of measure	Unit cost
2389	Shin guards	1	2	4.99
2570	Socks	2	2	3.90
3712	Boots	1	2	47.99

- 5 Using the sample of first normal form compliant data in the following table, adapt it to be compliant to second normal form.

ORDER#	ORDER DATE	BATTERY CODE	BATTERY DESCRIPTION	QUANTITY	UNIT COST	CUSTOMER ID	FIRST NAME	LAST NAME	PHONE CONTACT	ADDRESS	SUBURB	POSTCODE
234	3/03/2018	AA	1.5V Size AA battery	10	1.5	1	John	Smith	041111111	26 Charge St	Tin Can Bay	4580
234	3/03/2018	C	1.5V Size C battery	4	2.4	1	John	Smith	041111111	26 Charge St	Tin Can Bay	4580
235	3/04/2018	9V	9V Size battery	8	4.2	1	John	Smith	041111111	26 Charge St	Tin Can Bay	4580
236	5/04/2018	AA	1.5V Size AA battery	2	1.5	2	Hilary	Jones	042121212	28 Amp Road	Rainbow Beach	4581
237	7/04/2018	AA	1.5V Size AA battery	11	1.5	2	Hilary	Jones	042121212	28 Amp Road	Rainbow Beach	4581
237	7/04/2018	9V	9V Size battery	18	4.2	2	Hilary	Jones	042121212	28 Amp Road	Rainbow Beach	4581

- 6 Using the sample data in these second normal form compliant tables, adapt it to be compliant to third normal form.

ANIMAL ID	ANIMAL NAME	ANIMAL TYPE	ANIMAL BREED	TAGCODE	OWNER ID
1	Bob	Bird	Budgerigar	88TTZ66K	1
2	Lilly	Cat	Persian	43PEJ23B	3
3	Smiggle	Dog	Fox terrier	28QJQ11Z	2
4	Jake	Bird	Budgerigar	29UJN62Y	2
5	Cutey	Bird	Finch	43PED23A	5
6	Chalky	Dog	Poodle	68EFJ34Q	3
7	Terror	Cat	Siamese	98RFG34A	4
8	Apollo	Dog	Maltese	12EDA34F	5
9	Poppy	Dog	Beagle	45TYJ54Q	4

OWNER ID	OWNER NAME	PHONE	ADDRESS	SUBURB
1	Peter	(07) 4197 9320	22 Short St	Urangan
2	Ben	(07) 4197 9321	23 Long St	Point Vernon
3	Ken	(07) 4197 9321	23 Long St	Point Vernon
4	Mark	(07) 4197 9373	42 Hitchhiker Pl	Maryborough
5	Erin	(07) 4197 9340	23 April Lane	Gladstone

2.3

GENERATING DATA-DRIVEN APPLICATIONS AND SOLUTIONS

This section focuses on exploring the knowledge and skills required to generate data-driven digital solutions in a chosen technology context. Through the use of a case study web application and underlying relational database and database management system, we will investigate how relational schemas can be generated into a well-ordered database. We will also explore, practise and use Structured Query Language (SQL) syntax to create, modify, delete, store and retrieve data and generate database structures to solve data-driven digital solutions. Finally, we will explore opportunities to enact algorithms involving SQL calls to be implemented in different technology contexts using available programming environments and relational database management systems.

Working with relational schemas

Relational schemas are the completed diagrammatic representation of the ready-to-be-implemented relational database structure. While they do not contain every facet of information relating to implementing the database, the key components such as tables, fields, basic constraints, keys and referential links are contained in a thorough and complete relational schema.

To consider relational schemas, we will use the Garden Express online web store database case study.

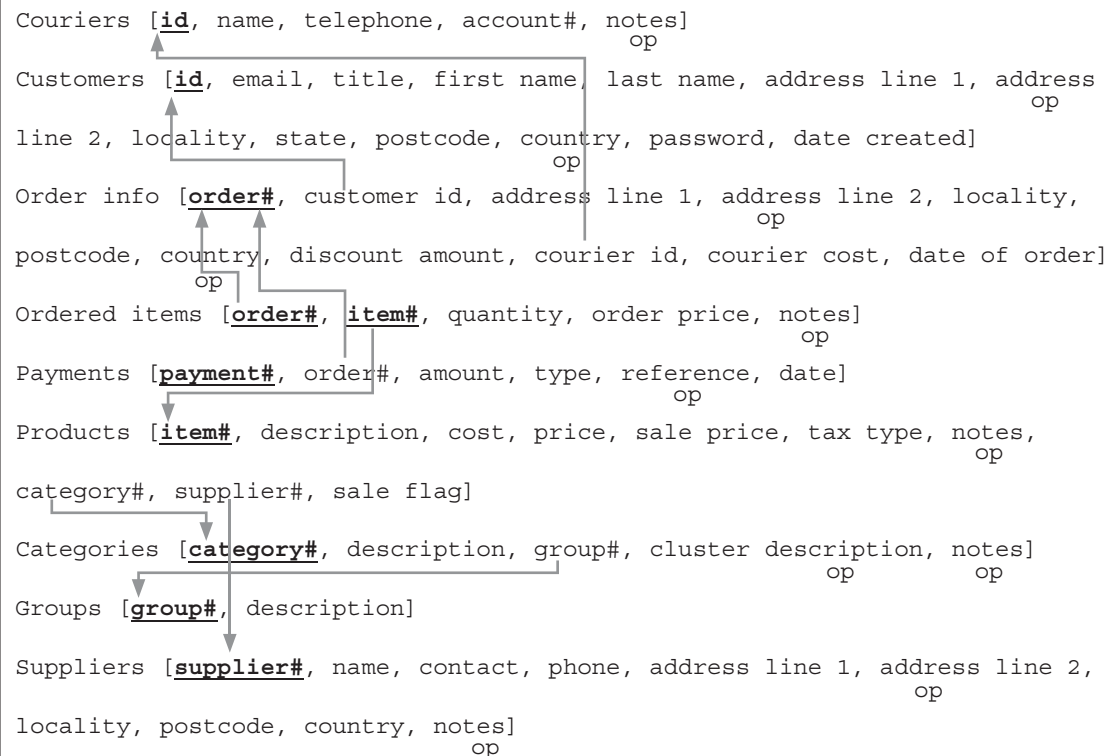


Figure 2.3.1 Sample relational schema for the Garden Express database

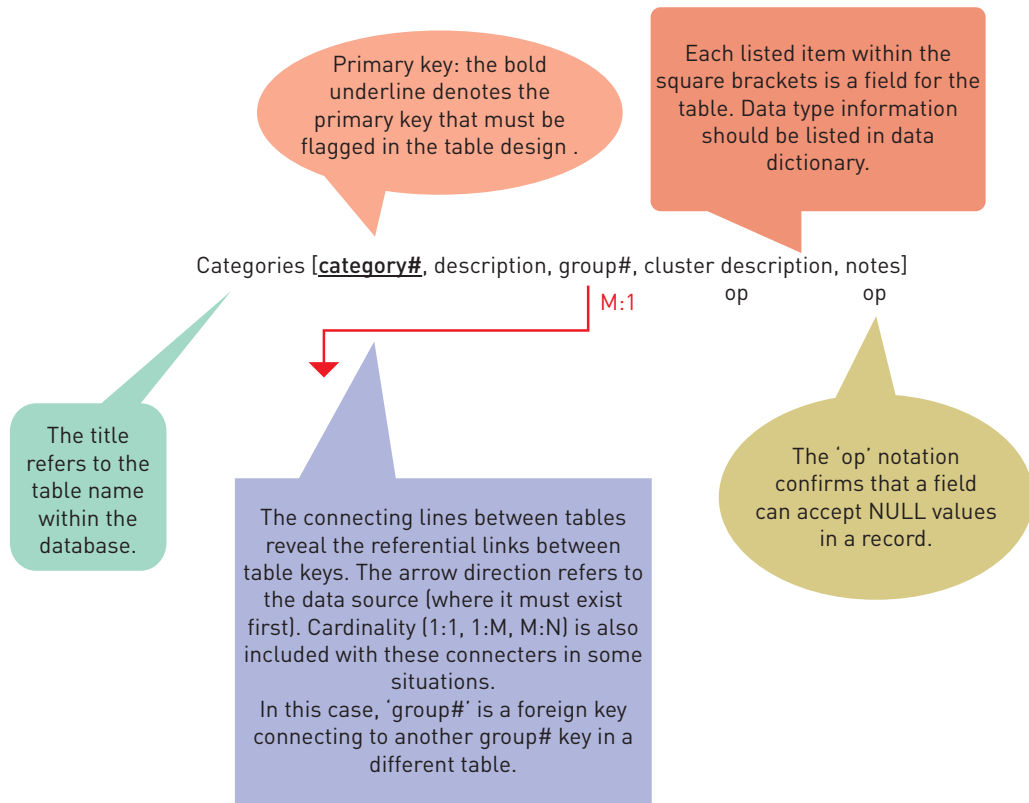


Figure 2.3.2 Features of a relational schema

Figure 2.3.2 reviews the features of a relational schema. These identified features are used as a generation plan within the database. When creating tables, try to include all required features and constraints to table structures before adding rows of data. It can be challenging to make extensive alterations to table structures. In some cases, a table may need to be deleted and re-created. This is problematic if the database now contains values.

Implementing a relational schema

Once you have gained the skills to interpret a relational schema and its associated features, it is time to implement it in a database management system. While each database management system is different, common processes exist in implementing a database. In this walkthrough, a PostgreSQL database will be created using the Database .NET visual editor. In the next section you will also be shown how to implement tables via DDL (Data Definition Language) SQL. In this example, a table will be created using the visual editor (see figure 2.3.3a).

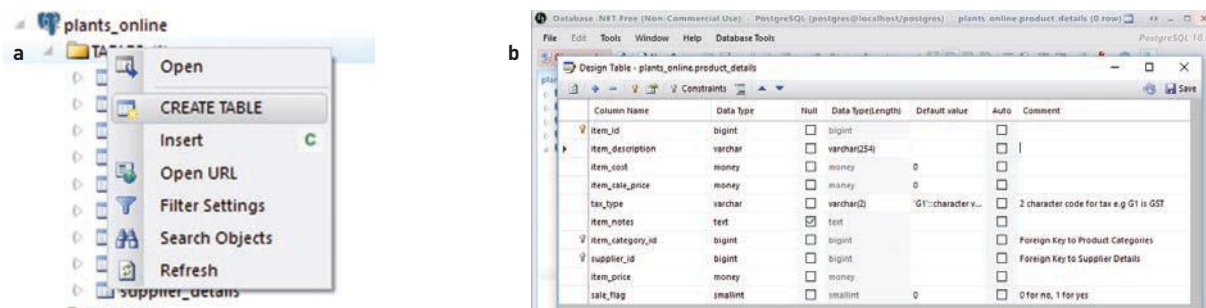


Figure 2.3.3 a, b Creating a table in the Database .NET visual editor

In figure 2.3.3b you can see how table fields are listed along with their appropriate data types. Data type information should be drawn from the data dictionary associated with the relational schema. To indicate a primary key, right-click on the field item to select it from the context menu. Set the NULL check boxes for those fields that can accept a NULL value.

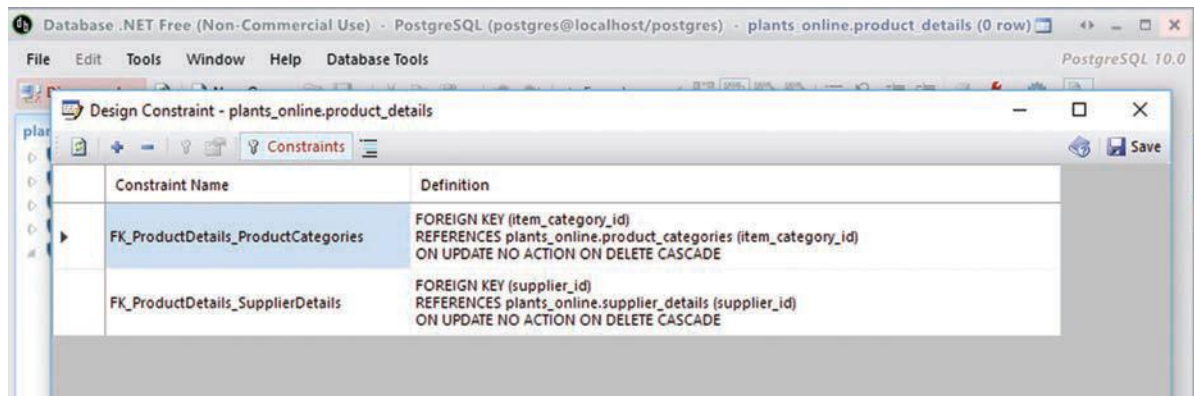


Figure 2.3.4 Defining foreign key constraints helps to maintain referential integrity.

By right-clicking on foreign key fields, referential constraints can be added to link table data together and inform referential integrity of the relational database system (figure 2.3.4).

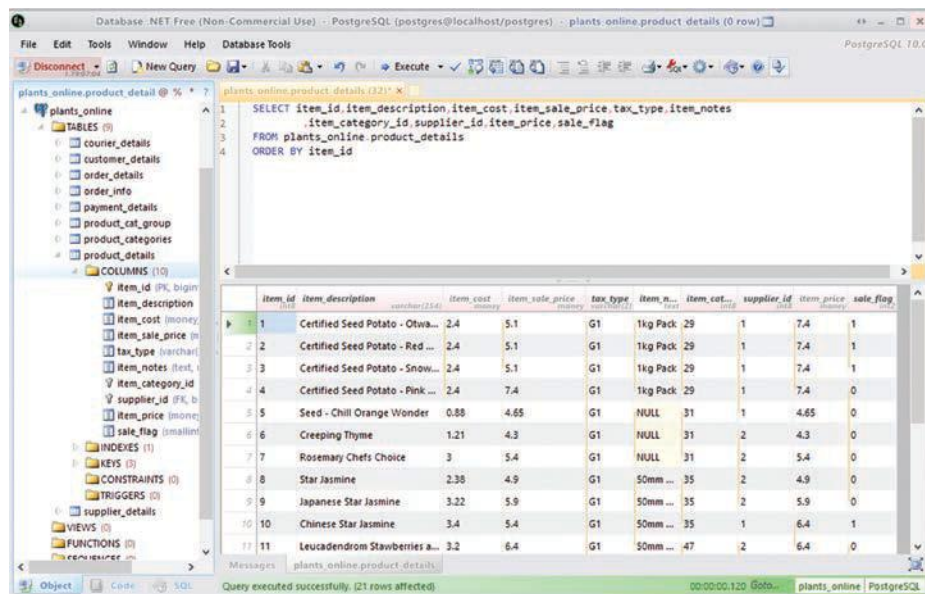


Figure 2.3.5 In Database .NET, data can be entered via the visual editor or with SQL commands.

Once all tables, features and constraints have been implemented, sample data can be entered via the visual editor (figure 2.3.5). In the next section, you will explore Structured Query Language and its capabilities to define and manipulate a relational database such as this.

Working with data: Structured Query Language (SQL)

One of the most common ways to manipulate and integrate data from relational databases is through the use of Structured Query Language (SQL). SQL is a highly evolved language used for managing data stored in relational databases, and is executed via a relational database management system. It plays a central role in the



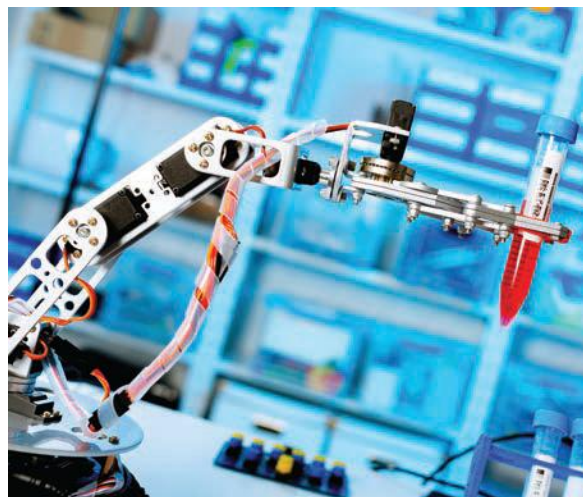
iStock Editorial/Getty Images Plus/Katharina13



iStock Unreleased/Getty Images Plus/hocus-focus



Getty Images/Hugh Peterswald/Pacific Press/LightRocket



Shutterstock.com/science photo

Figure 2.3.6 SQL is an important part of most data-driven digital solutions.

creation, changing, disposal and manipulation of data in data-driven applications systems and is at the heart of most of the digital tools we use every day. SQL is popular because it adheres to standards and its syntax is simple and uncomplicated. Many variants of SQL have been widely adopted by software vendors across the world.

The development of this database language came about as a result of IBM researcher Edgar Codd's work in defining the concept of the relational database model in the late 1960s and early 1970s. At that time, IBM researchers developed early versions of SQL, which was initially referred to as SEQUEL (Structured English Query Language) before the name was later shortened to SQL. To this day, 'SQL' is widely pronounced 'sequel' by those in the industry, and the vendor-specific product Microsoft SQL Server is referred to as 'sequel' server.

SQL comes in many variants, depending on the platform and the environment in which it is being used. The American National Standards Institute (ANSI) has defined the standard base version of SQL. ANSI SQL is supported by most database management systems but DBMS vendor-specific extensions to the standard syntax are also available. This section will cover only the core elements of ANSI SQL. This baseline version will be used to explore and navigate the application of SQL as well as the completion of environment-independent learning exercises.

SQL variants in database management systems

When it is time to implement SQL in project work, more than likely it will involve an SQL variant. The following table provides a snapshot of some common database management servers and their associated SQL language. You may need to revisit this table after learning SQL so you can understand the examples listed.

Table 2.3.1 SQL variants in common database management systems

DBMS	SQL VARIANT	EXAMPLE DIFFERENCE
MySQL Server	MySQL	SELECT columns FROM tablename ORDER BY column ASC LIMIT n
Microsoft SQL Server	T-SQL	SELECT TOP n columns FROM tablename ORDER BY key ASC
Microsoft Access	Jet SQL	SELECT TOP n columns FROM tablename ORDER BY key ASC
Oracle	PL/SQL	The main difference is that it combines programming elements.
PostgreSQL	PL/pgSQL	Closely ANSI SQL compliant
SQLite	SQLite	Generally ANSI SQL with minor feature omissions – e.g. RIGHT OUTER JOIN/FULL OUTER JOIN

2.3.2 SQLite
2.3.3 Microsoft SQL
2.3.4 SQL Server
Management Studio
2.3.5 Oracle
2.3.6 PostgreSQL
2.3.7 Fishcode
2.3.8 AWS
Free trial
2.3.9 Installation
guides

PostgreSQL and Database .NET

This unit learning sequence is based around PostgreSQL. PostgreSQL is a fully open-source relational database management system that is free to use and will run across different platforms. It has been chosen for this learning sequence due to its extensive compatibility and compliance with ANSI SQL. While it can be installed locally on an individual device, it can also be used on a web server or accessed remotely in a cloud host scenario.

Database .NET is a database management visual editor tool that can interface with a wide array of relational database management systems. Using a viable visual editor for SQL is a productive and accessible way to learn the features of the language and to explore the inner features, structures, keys and constraints of an implemented database. Tools such as Database .NET will also assist in creating new databases and implementing data integrity constraints.

If PostgreSQL is not available or cannot be installed in your environment, SQLite would also be another viable choice for completing this learning sequence.

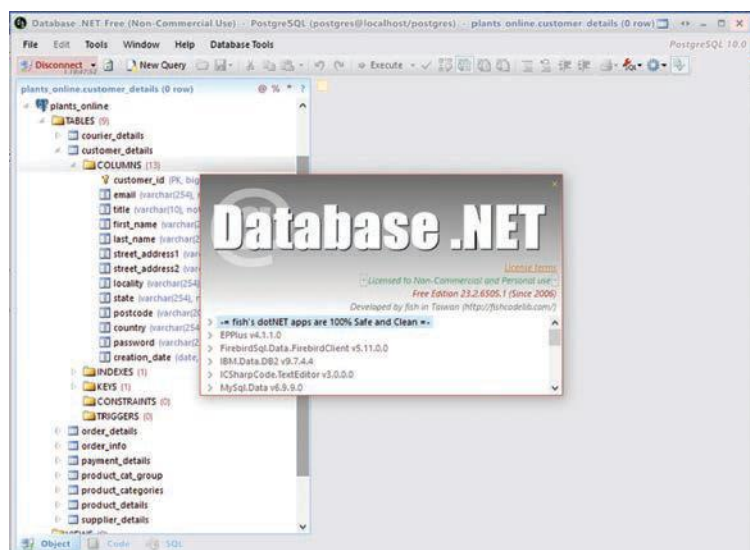


Figure 2.3.7 The Database .NET visual editor supports many different versions of SQL.

Features of SQL

Structured Query Language (SQL) plays a core role in most relational database management systems. Most SQL statements are classified into two groups: data definition and data manipulation.

Table 2.3.2 Structured Query Language (SQL) features

DATA DEFINITION LANGUAGE (DDL) COMMANDS	DATA MANIPULATION LANGUAGE (DML) COMMANDS
CREATE	SELECT
ALTER	INSERT
DROP	UPDATE
<i>TRUNCATE</i>	DELETE
<i>RENAME</i>	<i>MERGE</i>
	<i>LOCK</i>
TRANSACTION CONTROL LANGUAGE (TCL) COMMANDS	DATA CONTROL LANGUAGE (DCL) COMMANDS
COMMIT	GRANT
ROLLBACK	REVOKE

While there are a wide set of SQL features available, this course covers the core elements of data definition (databases and tables) and data manipulation (select, insert, update, delete). Those SQL features shown in *italics* in table 2.3.2 are additional areas that can be investigated to gain an in-depth understanding of the capabilities of advanced SQL functionality.

SQL Data Definition Language (DDL)

SQL DDL statements are used to create, alter or remove structural components of a database. This includes the creation and removal of the database itself, table structures, keys and other constraint settings.

CREATE command

The CREATE command provides a mechanism to establish databases, as well as tables and other objects within a database. For creating a table, the basic syntax is as shown in figure 2.3.8.

<pre>CREATE TABLE table_name (field1 datatype [CONSTRAINTS] [PRIMARY KEY], field2 datatype [CONSTRAINTS] [PRIMARY KEY], field3 datatype [CONSTRAINTS] [PRIMARY KEY], ... fieldN datatype [CONSTRAINTS] [PRIMARY KEY], FOREIGN KEY (fieldX) REFERENCES foreign_table (foreign_key_field)</pre>	Constraints: eg. NOT NULL (optional) IDENTIFYING PRIMARY KEY (optional) FOREIGN KEY declaration for links between tables

Figure 2.3.8 SQL CREATE syntax

The following SQL code snippets are examples from the Garden Express web store database.

```
CREATE TABLE plants_online.customer_details (
  customer_id bigint NOT NULL PRIMARY KEY,
  email varchar(254) NOT NULL,
  title varchar(10) NOT NULL,
  first_name varchar(254) NOT NULL,
  last_name varchar(254) NOT NULL,
  street_address1 varchar(254) NOT NULL,
```

```

street_address2 varchar(254),
locality varchar(254) NOT NULL,
state varchar(254) NOT NULL,
postcode varchar(20) NOT NULL,
country varchar(254),
password varchar(254) NOT NULL,
creation_date date NOT NULL
);

```

This CREATE TABLE statement establishes the core features of the ‘customer_details’ table. It should be noted that ‘street_address2’ and ‘country’ fields are optional and can contain NULL values due to no constraint having been applied to them.

```

CREATE TABLE plants_online.product_categories (
  item_category_id bigint NOT NULL PRIMARY KEY,
  item_category_desc varchar(254) NOT NULL,
  cat_group_id bigint NOT NULL,
  cluster_category_desc varchar(254),
  item_category_notes varchar(254),
  FOREIGN KEY (cat_group_id)
    REFERENCES plants_online.product_cat_group (cat_group_id)
);

```

This CREATE TABLE statement establishes the core features of the ‘product_categories’ table. In this example, a foreign key reference to an existing table (product_cat_group) is declared to enable relational referential integrity constraints between the tables. In this case, if a user attempted to enter a record into this (product_categories) table where the ‘cat_group_id’ value did not exist in the referenced (product_cat_group) table, the database management system would refuse to commit the data. By adding foreign key references, database designers and administrators can ensure referential integrity of the database.

The CREATE command can also be used to create the database. While this is the case, the use of a SQL editor tool can assist with this in a graphical and easy-to-use manner.

ALTER command

The ALTER command provides a mechanism to edit databases, as well as tables and other objects within a database. There are many options available for altering a table. Here are six scenarios for altering a table.

To add an extra field (column)	ALTER TABLE table_name ADD field_name datatype;
To remove a field (column)	ALTER TABLE table_name DROP COLUMN field_name;
To add a primary key	ALTER TABLE table_name ADD CONSTRAINT table_primary_key PRIMARY KEY (field1, field2...);
To rename a table	ALTER TABLE name RENAME TO new_name;
To rename a field (column)	ALTER TABLE name RENAME field_name TO new_field_name
To set a field constraint as NOT NULL	ALTER TABLE table_name MODIFY field_name datatype NOT NULL;



The following SQL code snippets are examples from the Garden Express webstore database.

```
ALTER TABLE plants_online.product_categories
  MODIFY item_category_notes varchar(254) NOT NULL;
```

This ALTER TABLE statement modifies the constraints of the 'item_category_notes' field by determining that it must contain a value. This command would fail to commit to the database if NULL values already existed in records. NULL values in existing records would need to be updated before successfully enacting this SQL statement.

```
ALTER TABLE plants_online.customer_details
  ADD loyalty_card_number varchar(20);
```

This ALTER TABLE statement adds an additional field to the 'customer_details' table. The initial value used for all existing rows will be NULL. Some SQL variants allow a default value to be specified when adding a field to a table. In ANSI SQL, if a NOT NULL constraint is required, it must be added only after data has been added to the new column.

DROP command

The DROP command provides a mechanism to delete databases as well as tables and other objects from within a database. The syntax for dropping a table or database is shown below.

To DROP a database	<code>DROP DATABASE database_name;</code>
To DROP a table	<code>DROP TABLE table_name;</code>

It should be noted that this command deletes the database or table instance and therefore should be used carefully. The following SQL code snippets are examples from the Garden Express web store database.

The following DROP TABLE statement will completely delete the 'product_categories' table and all data contained within it:

```
DROP TABLE plants_online.product_categories;
```

The following DROP DATABASE statement will completely delete the entire Garden Express web store database from the database management system:

```
DROP DATABASE plants_online;
```

SQL Data Manipulation Language (DML)

Data manipulation language elements are the most commonly used features of SQL. Programmers and database administrators use these commands to retrieve a range of datasets as well as to modify field values and whole records within data tables. This exploration of SQL will include basic to complex SELECT statements, including retrieving data across multiple tables, applying a range of conditions and using data to produce grouped and calculated summaries live from the database. Completing these tasks via a database means that users receive up-to-date live data and can make developed systems more useful and accurate.

When granting access to users of a database, several levels of command access should be considered. This would be one way to avoid accidental deletion or corruption of the database and of the overall data-driven system. Generic accounts that are used by the overall system for database entry should also be given strict access only to those SQL command features required.



2.3.11 SQL
information

SELECT statement

SELECT statements are at the heart of retrieving data from a database and are the most commonly used type of SQL statement.

The basis for the SELECT syntax is as follows:

SELECT fields

FROM table

The following example demonstrates extracting data from the 'product_details' table:

```
SELECT item_description
FROM plants_online.product_details
```

The following example can be extended to include multiple fields. In this example, the item description, cost price and supplier code are all selected.

```
SELECT item_description, item_cost, supplier_id
FROM plants_online.product_details;
```

It should be noted that table notation is used as standard in PostgreSQL. This means that in every SQL statement, a table is referred to formally as 'DATABASENAME.TABLENAME'. Some SQL variants do not require this and will work without it, but in the case of PostgreSQL it is required.



ITEM_DESCRIPTION
100 Days of Colour Collection
Landscape Bargain College – 400 Bulbs
Tulip Black and Yellow Blend
Tulip Triumph Ile de France
Black Mondo Grass
Dwarf Green Mondo Grass
Copper Shield Fern
Emerald Queen Fern
Protea Stately (PBR)
Protea – Possum Magic
Leucadendron Strawberries and Cream
Chinese Star Jasmine
Japanese Star Jasmine
Star Jasmine
Rosemary Chefs Choice
Creeping Thyme
Seed – Chill Orange Wonder
Certified Seed Potato – Pink Fir Apple 1kg
Certified Seed Potato – Snow Gem 1kg
Certified Seed Potato – Red Norland 1kg
Certified Seed Potato – Otway Red 1kg

ITEM_DESCRIPTION	ITEM_COST	SUPPLIER_ID
100 Days of Colour Collection	22.53	1
Landscape Bargain College – 400 Bulbs	23	1
Tulip Black and Yellow Blend	12.72	1
Tulip Triumph Ile de France	4.5	1
Black Mondo Grass	3.33	2
Dwarf Green Mondo Grass	2.11	2
Copper Shield Fern	1.88	1
Emerald Queen Fern	1.88	2
Protea Stately (PBR)	5.83	2
Protea – Possum Magic	5.83	2
Leucadendron Strawberries and Cream	3.2	2
Chinese Star Jasmine	3.4	1
Japanese Star Jasmine	3.22	2
Star Jasmine	2.38	2
Rosemary Chefs Choice	3	2
Creeping Thyme	1.21	2
Seed – Chill Orange Wonder	0.88	1
Certified Seed Potato – Pink Fir Apple 1kg	2.4	1

ITEM_DESCRIPTION	ITEM_COST	SUPPLIER_ID
Certified Seed Potato – Snow Gem 1kg	2.4	1
Certified Seed Potato – Red Norland 1kg	2.4	1
Certified Seed Potato – Otway Red 1kg	2.4	1

Selecting all fields or using wildcards

There are two ways to include all fields in a SELECT statement. A statement can manually include all fields in a table. Another method is to use the 'select all' wildcard symbol (*).

```
SELECT order_id, item_id, qty, unit_order_price, item_order_notes
FROM plants_online.order_details
or
SELECT *
FROM plants_online.order_details
```

ORDER_ID	ITEM_ID	QTY	UNIT_ORDER_PRICE	ITEM_ORDER_NOTES
1	4	1	7.22	NULL
1	11	4	6.44	NULL
1	12	11	9.23	NULL
1	15	3	5.33	NULL
2	4	20	8.33	NULL
2	9	2	23.48	purple colour

Conditions (the WHERE clause)

In most situations, data administrators and programmers need to filter data, rather than retrieve a complete data dump from a table. The addition of the WHERE clause enables this. It allows conditions to be tested on data searches, and only those records that match these requirements are returned.

For example, we may wish to find all customers with the last name 'Smith'.

```
SELECT customer_id, email, title, first_name, last_name
FROM plants_online.customer_details
WHERE last_name = 'Smith'
```

CUSTOMER_ID	EMAIL	TITLE	FIRST_NAME	LAST_NAME
1	Harry@gtcq.com.au	Mr	Harry	Smith

Another scenario would be to find a list of product categories associated with category group 2 (Edible Produce).

```
SELECT *
FROM plants_online.product_categories
WHERE cat_group_id = 2
```

ITEM_CATEGORY_ID	ITEM_CATEGORY_DESC	CAT_GROUP_ID	CLUSTER_CATEGORY_DESC	ITEM_CATEGORY_NOTES
28	Berries and Currants	2	NULL	NULL
29	Certified Seed Potatoes	2	NULL	NULL
30	Fruiting Vines	2	NULL	NULL
31	Herbs, Chillis and Spices	2	NULL	NULL
32	Other Produce Plants	2	NULL	NULL

An example of a condition developed using NULL is:

```
SELECT *
FROM plants_online.product_categories
WHERE item_category_notes IS NULL
```

Some fields in a relational database table are allowed to be empty. SQL refers to this as a NULL value. It is possible to develop a condition using NULL.



Note that 'IS' is used instead of '=' in this situation.

Advanced WHERE conditions

In many circumstances, data administrators and programmers need to use SQL to refine data selection even further. Through the use of logical operators (AND, OR and NOT) and comparison operators (<, <=, >, >=, <>) programmers can develop an array of possible solutions to complex data manipulation problems.

By incorporating logical operators into the WHERE condition statement, programmers are able to develop SQL solutions that can meet multiple conditions. For example, in the previous example, product categories were selected where the condition of a specific group was met. In the example below, an additional constraint has been added to also filter out unwanted records that are not part of the 'Spring Flowering' group.

```
SELECT *
FROM plants_online.product_categories
WHERE cat_group_id = 1 AND cluster_category_desc = 'Spring Flowering'
```

ITEM_CATEGORY_ID	ITEM_CATEGORY_DESC	CAT_GROUP_ID	CLUSTER_CATEGORY_DESC	ITEM_CATEGORY_NOTES
1	Daffodils	1	Spring Flowering	NULL
2	Hyacinths	1	Spring Flowering	NULL
3	Tulips	1	Spring Flowering	NULL
4	Spring Bulb Collections	1	Spring Flowering	NULL
5	Allium	1	Spring Flowering	NULL
6	Anemones	1	Spring Flowering	NULL
7	Baby Gladioli	1	Spring Flowering	NULL
8	Dutch Crocus	1	Spring Flowering	NULL
9	Dutch Iris	1	Spring Flowering	NULL
10	Dwarf Iris	1	Spring Flowering	NULL
11	Freesias	1	Spring Flowering	NULL

CASE STUDY

WEB CONTEXT EXAMPLE: GARDEN EXPRESS (CONTINUED FROM PAGE 145)

The SQL SELECT statement used in the example here is a practical real-life example of SQL use in the Garden Express online shopping web store. Then, in specific sections of the menu system, categories must be presented so that they collectively relate to a specific category group (cat_group_id) and a collective subcategory listing (cluster_category_desc).

Behind the developed code in this site, an SQL SELECT query similar to this statement (below) would produce data to populate the given menu.

```
SELECT item_category_desc
FROM plants_online.product_categories
WHERE cat_group_id = 1 AND cluster_category_desc = 'Spring
Flowering'
```



Figure 2.3.9
Garden Express
drop-down menu

Garden Express Australia Ltd

Logical operators

Logical operator usage is based on an order of precedence (importance). Parentheses (brackets) can also be used alongside the logical operators to isolate order of precedence and allow for a range of condition possibilities. The order of precedence for parentheses and logical operators is:

Element	Logical operator	Precedence
Parentheses (brackets)	--	1
--	NOT	2
--	AND	3
--	OR	4

item_category_desc
Daffodils
Hyacinths
Tulips
Spring Bulb Collections
Allium
Anemones
Baby Gladioli
Dutch Crocus
Dutch Iris
Dwarf Iris
Freesias

ACTIVITY 2.3.1

- 1 What would the following WHERE condition produce?

```
SELECT item_category_desc
FROM plants_online.product_categories
WHERE (cat_group_id = 1 AND cluster_category_desc = 'Spring Flowering')
OR (cat_group_id = 2 AND cluster_category_desc IS NULL)
```

- 2 Would the following WHERE condition produce a result? Why or Why not?

```
SELECT item_id, item_description
FROM plants_online.product_details
WHERE (item_notes IS NULL AND item_cost < '10.2') AND (NOT supplier_id = 1
AND item_cost > '10.2')
```

The OR operator can be used to produce a list of products supplied by supplier_id '1' or those items that are flagged as being on sale.

```
SELECT item_id, item_description, item_cost, supplier_id, sale_flag
FROM plants_online.product_details
WHERE supplier_id = 1 OR sale_flag = 1
```





item_id	item_description	item_cost	supplier_id	sale_flag
2	Certified Seed Potato – Red Norland 1kg	2.4	1	1
1	Certified Seed Potato – Otway Red 1kg	2.4	1	1
3	Certified Seed Potato – Snow Gem 1kg	2.4	1	1
4	Certified Seed Potato – Pink Fir Apple 1kg	2.4	1	0
5	Seed – Chill Orange Wonder	0.88	1	0
10	Chinese Star Jasmine	3.4	1	1
14	Emerald Queen Fern	1.88	2	1
15	Copper Shield Fern	1.88	1	0
17	Black Mondo Grass	3.33	2	1
18	Tulip Triumph Ile de France	4.5	1	0
19	Tulip Black and Yellow Blend	12.72	1	1
20	Landscape Bargain College – 400 Bulbs	23	1	1
21	100 Days of Colour Collection	22.53	1	1

Comparison operators

Comparison operators provide opportunities to incorporate conditions other than 'is equal to' (=).

An example might be that you require a list of all sale items with an item cost that is more than or equal to \$3.00.

```
SELECT item_id,item_description,item_cost,supplier_id,sale_flag
FROM plants_online.product_details
WHERE sale_flag = 1 AND item_cost >= '3.00'
```

item_id	item_description	item_cost	supplier_id	sale_flag
10	Chinese Star Jasmine	3.4	1	1
17	Black Mondo Grass	3.33	2	1
19	Tulip Black and Yellow Blend	12.72	1	1
20	Landscape Bargain College – 400 Bulbs	23	1	1
21	100 Days of Colour Collection	22.53	1	1

Partial matches

It is also possible to add conditions based on partial information using the LIKE clause. The LIKE clause replaces the comparison operator and is used in conjunction with two possible wildcards: % and _.

The '%' wildcard allows for 0..N possibilities and the '_' wildcard matches exactly one character. The following example finds all items that begin with 'B'.

```
SELECT item_id,item_description,item_cost,supplier_id,sale_flag
FROM plants_online.product_details
WHERE item_description LIKE 'B%'
```

item_id	item_description	item_cost	supplier_id	sale_flag
17	Black Mondo Grass	3.33	2	1

The following example specifically filters for any item description that has the character 'B' contained within it.

```
SELECT item_id,item_description,item_cost,supplier_id,sale_flag
FROM plants_online.product_details
WHERE item_description LIKE '%B%'
```

item_id	item_description	item_cost	supplier_id	sale_flag
11	Leucadendron Stawberries and Cream	3.2	2	0
13	Protea Stately (PBR)	5.83	2	0
17	Black Mondo Grass	3.33	2	1
19	Tulip Black and Yellow Blend	12.72	1	1
20	Landscape Bargain College - 400 Bulbs	23	1	1

The '_' wildcard can also be incorporated into a LIKE clause. It denotes a single (1) wildcard character in its place. The following example filters item category description to start with the letter 'O', then accepts anything as the second character, specifies the third character as 'h' and then accepts anything beyond that.

```
SELECT item_category_desc
FROM plants_online.product_categories
WHERE item_category_desc LIKE 'O_h%'
```

item_category_desc
Other Autumn Bulbs
Other Produce Plants
Other Climbing Varieties
Other Shrubs Varieties
Other Foliage Varieties
Other Grasses and Strap Leafed Plants

Table joins

As previously explored, relational databases separate data across different tables to significantly reduce redundancy and improve data integrity and reliability. However, in most real-life uses of presenting data, it must be harvested collectively from different tables to give end users complete meaning. Table joins in SQL allow for this functionality. They allow SQL SELECT statements to gather fields from multiple tables and join data records for meaningful use.

Let's look at the example where item details and category descriptions are needed in one SQL SELECT statement.

```
SELECT item_id, item_description, item_category_desc
FROM plants_online.product_details, plants_online.product_categories
```

This produces a valid SQL statement but, if you look closely at the output of data, you will see that something is wrong.

item_id	item_description	item_category_desc
1	Certified Seed Potato - Otway Red 1kg	Daffodils
1	Certified Seed Potato - Otway Red 1kg	Hyacinths
1	Certified Seed Potato - Otway Red 1kg	Tulips
1	Certified Seed Potato - Otway Red 1kg	Spring Bulb Collections
1	Certified Seed Potato - Otway Red 1kg	Allium
1	Certified Seed Potato - Otway Red 1kg	Anemones
1	Certified Seed Potato - Otway Red 1kg	Baby Gladioli
1	Certified Seed Potato - Otway Red 1kg	Dutch Crocus





item_id	item_description	item_category_desc
1	Certified Seed Potato – Otway Red 1kg	Dutch Iris
1	Certified Seed Potato – Otway Red 1kg	Dwarf Iris
1	Certified Seed Potato – Otway Red 1kg	Freesias
1	Certified Seed Potato – Otway Red 1kg	Arum Lillies
1	Certified Seed Potato – Otway Red 1kg	Tuberous Begonias
1	Certified Seed Potato – Otway Red 1kg	Calla Lillies
1	Certified Seed Potato – Otway Red 1kg	Dahlia Ball
1	Certified Seed Potato – Otway Red 1kg	Dahlia Cacti
1	Certified Seed Potato – Otway Red 1kg	Dahlia Dark Foliage
1	Certified Seed Potato – Otway Red 1kg	Dahlia Decorative
1	Certified Seed Potato – Otway Red 1kg	Dahlia Delbard
...	...	...
2	Certified Seed Potato – Red Norland 1kg	Daffodils
2	Certified Seed Potato – Red Norland 1kg	Hyacinths
2	Certified Seed Potato – Red Norland 1kg	Tulips
2	Certified Seed Potato – Red Norland 1kg	Spring Bulb Collections
2	Certified Seed Potato – Red Norland 1kg	Allium
2	Certified Seed Potato – Red Norland 1kg	Anemones
2	Certified Seed Potato – Red Norland 1kg	Baby Gladioli
2	Certified Seed Potato – Red Norland 1kg	Dutch Crocus
2	Certified Seed Potato – Red Norland 1kg	Dutch Iris

(1365 records)

The problem is that there is no correct linking occurring between table records. In this example, item_id '1' should link to the 'Certified Potatoes' product category, as item_category_id exists in both tables and the values in each should link the records together. Instead, the example above has given us every possible combination between the records in the product_details table and the product_categories table. This is referred to a Cartesian product and does not represent total valid data. It is therefore important in table joins statements to filter out any invalid combinations of data in order to only retrieve key value linked records.

SQL allows two methods to achieve this type of table join. Here are the two possible SQL SELECT statements to produce the filtered set of data in the table below. In both examples, when field names exist in multiple tables, table.column notation must be used to distinguish which field is being referred to.

```
SELECT item_id, item_description, item_category_desc
FROM plants_online.product_details INNER JOIN plants_online.product_categories
ON product_details.item_category_id = product_categories.item_category_id
```

or

```
SELECT item_id, item_description, item_category_desc
FROM plants_online.product_details, plants_online.product_categories
WHERE product_details.item_category_id = product_categories.item_category_id
```



item_id	item_description	item_category_desc
1	Certified Seed Potato – Otway Red 1kg	Certified Seed Potatoes
2	Certified Seed Potato – Red Norland 1kg	Certified Seed Potatoes
3	Certified Seed Potato – Snow Gem 1kg	Certified Seed Potatoes
4	Certified Seed Potato – Pink Fir Apple 1kg	Certified Seed Potatoes
5	Seed – Chill Orange Wonder	Herbs, Chillis and Spices
6	Creeping Thyme	Herbs, Chillis and Spices
7	Rosemary Chefs Choice	Herbs, Chillis and Spices
8	Star Jasmine	Jasmine
9	Japanese Star Jasmine	Jasmine
10	Chinese Star Jasmine	Jasmine
11	Leucadendron Stawberries and Cream	Proteaceae
12	Protea – Possum Magic	Proteaceae
13	Protea Stately (PBR)	Proteaceae
14	Emerald Queen Fern	Ferns
15	Copper Shield Fern	Ferns
16	Dwarf Green Mondo Grass	Mondo Grass
17	Black Mondo Grass	Mondo Grass
18	Tulip Triumph Ile de France	Tulips
19	Tulip Black and Yellow Blend	Tulips
20	Landscape Bargain College – 400 Bulbs	Spring Bulb Collections
21	100 Days of Colour Collection	Spring Bulb Collections

As shown in the data projection, there is now valid joining of records due to key constraints now acting on each table.

In the next example, the previous SQL statement is modified to include a third table. Further variations of this example may be used to assist in generating an order form or invoice.

```
SELECT order_id, product_details.item_id, item_description, item_category_desc,
       qty, unit_order_price
FROM plants_online.product_details, plants_online.product_categories, plants_
     online.order_details
WHERE product_details.item_category_id = product_categories.item_category_id
AND product_details.item_id = order_details.item_id
```

order_id	item_id	item_description	item_category_desc	qty	unit_order_price
1	4	Certified Seed Potato – Pink Fir Apple 1kg	Certified Seed Potatoes	1	7.22
1	11	Leucadendrom Stawberries and Cream	Proteaceae	4	6.44
1	12	Protea – Possum Magic	Proteaceae	11	9.23
1	15	Copper Shield Fern	Ferns	3	5.33
2	4	Certified Seed Potato – Pink Fir Apple 1kg	Certified Seed Potatoes	20	8.33
2	9	Japanese Star Jasmine	Jasmine	2	23.48



If we explore the SQL statement in detail and colour-code which parts represent each table, it can be seen how data key links are enforced between tables.

```
SELECT order_id, product_details.item_id, item_description, item_category_desc,
       qty, unit_order_price
FROM plants_online.product_details, plants_online.product_categories, plants_
     online.order_details
WHERE product_details.item_category_id = product_categories.item_category_id
AND product_details.item_id = order_details.item_id
```

If we then compare those links with the table relationship diagram below, it can be seen that the SQL statement enforces links as per the intended relationship between the tables.

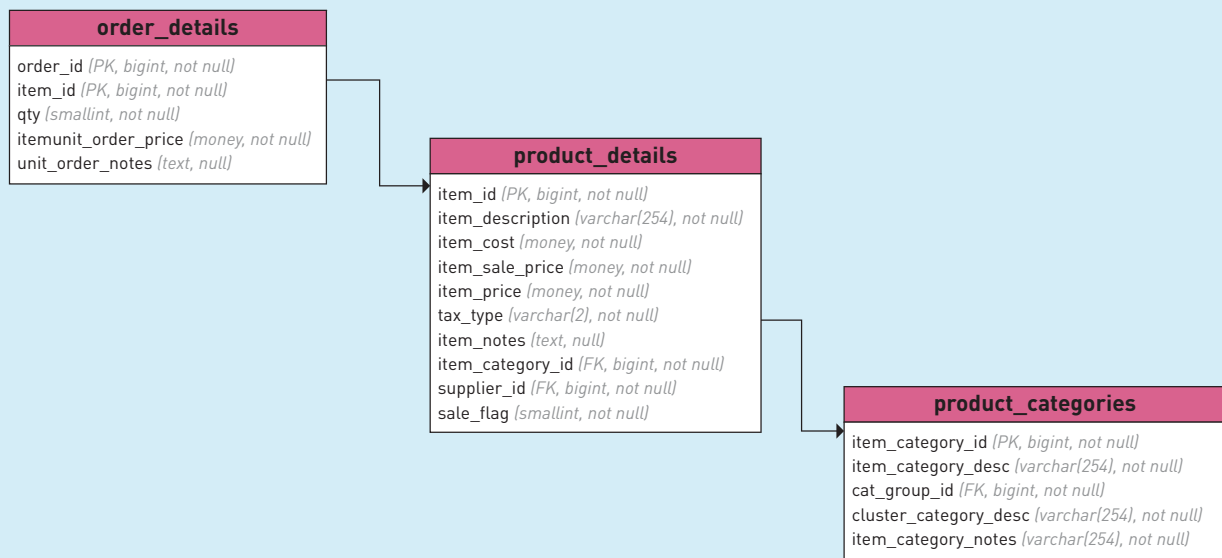


Figure 2.3.10 SQL joins require an understanding of the primary key/foreign key relationships between tables.

Therefore, when selecting data from multiple tables you should ensure that each table is linked by appropriate fields.

Calculations and functions

SQL provides capabilities to complete mathematical operations on the fly. This is an important feature as it can provide up-to-date results immediately that are continually reliable to end users as data is changed or added to over time. Data elements that can be calculated from base data do not need to be stored in the database. A prime example of calculated data that is not stored in a database is a person's age. Instead of storing age, databases tend to store a person's date of birth and complete an on-the-fly mathematical operation between the date of birth value and the current date value.

The core mathematical operators used by SQL are similar to those found in most text-based programming languages (see table 2.3.3).

Table 2.3.3 Mathematical operators used in SQL

OPERATION NAME	OPERATOR
Addition	+
Subtraction	-
Multiplication	*
Division	/
Modulus (remainder)	%
Exponential	^
Square root	√

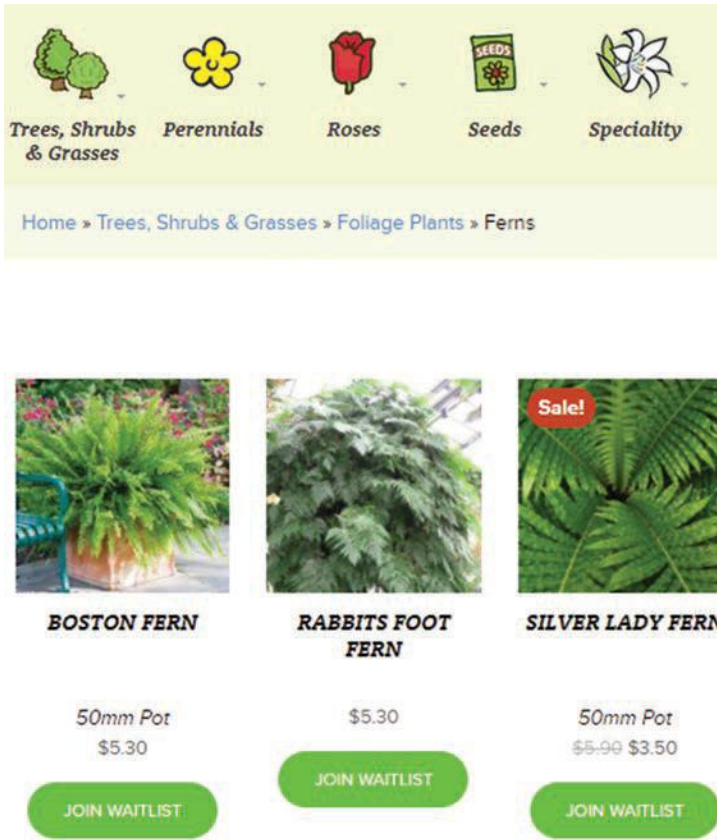


Figure 2.3.11 Detail of Garden Express web page showing sale price

ITEM_ID	ITEM_DESCRIPTION	ITEM_PRICE	ITEM_SALE_PRICE	SALE_SAVING
14	Emerald Queen Fern	5.90	3.50	2.40

Calculations could also be used in the previous three-table join example to create an `items_total` column on an order invoice. Note that calculations can appear as additional fields in the SQL result table using the field name defined after the keyword 'AS'.

```
SELECT order_id, product_details.item_id, item_description, item_category_desc,
       qty, unit_order_price, unit_order_price * qty AS items_total
FROM plants_online.product_details, plants_online.product_categories, plants_
     online.order_details
WHERE product_details.item_category_id = product_categories.item_category_id
AND product_details.item_id = order_details.item_id
```

ORDER_ID	ITEM_ID	ITEM_DESCRIPTION	ITEM_CATEGORY_DESC	QTY	UNIT_ORDER_PRICE	ITEMS_TOTAL
1	4	Certified Seed Potato – Pink Fir Apple 1kg	Certified Seed Potatoes	1	7.22	7.22
1	15	Copper Shield Fern	Ferns	3	5.33	15.99
2	4	Certified Seed Potato – Pink Fir Apple 1kg	Certified Seed Potatoes	20	8.33	166.6
2	9	Japanese Star Jasmine	Jasmine	2	23.48	46.96

In the 'product_details' table, some items are flagged as having a sale price. A suggestion from the Sales Manager at Garden Express is that a savings value could be displayed next to the sale price to encourage customers to take advantage of the product sales and assist in improving revenue to the company.

An SQL statement including the sale calculation could be used to solve this data problem.

```
SELECT item_id, item_
       description, item_price, item_
       sale_price, item_price - item_
       sale_price AS sale_saving
FROM plants_online.product_
     details
WHERE sale_flag = 1 AND item_
     id = 14
```


As shown in the SQL projection, the total for each quantity of products ordered has been calculated via the SQL statement.

ITEM_CATEGORY_ID	COMBINED_TEXT_FIELD
1	Daffodils - Spring Flowering
2	Hyacinths - Spring Flowering
3	Tulips - Spring Flowering
12	Arum Lillies - Summer Flowering
25	Lycoris - Autumn Flowering
34	Clematis - Climbing Plants
35	Jasmine - Climbing Plants
36	Wisteria - Climbing Plants

Just as with logical operators, brackets can be used within mathematical calculations to influence the order of precedence – for example: $(a+b)*(r-q)$ AS x.

It is also possible to concatenate (join) strings together in PostgreSQL using the '||' operator. An example is shown below.

```
SELECT item_category_id,
       item_category_desc || ' - ' ||
       cluster_category_desc AS
       combined_text_field
FROM plants_online.product_
categories
```

Aggregate functions are another method for completing calculations over large sets of data. In SQL, there is a core set of aggregate functions, including those in table 2.3.4.

Table 2.3.4 ANSI SQL aggregate functions

NAME	SQL CLAUSE	EXPLANATION
Average	AVG()	Calculates an average value across a set of selected data
Count	COUNT()	Counts the total number of records selected
Maximum	MAX()	Determines the highest value across a set of selected data
Minimum	MIN()	Determines the lowest value across a set of selected data
Sum	SUM()	Calculates a grand total of all values across a set of selected data

Mathematical operations calculate values for a row but aggregate functions calculate summary values for an entire column or group of rows. Aggregate functions, with the exception of COUNT(), will ignore NULL values when calculating results.

We will now explore some examples.

In the example below, the total number of order records is counted to produce a total figure.

```
SELECT COUNT(*) AS total_orders
FROM plants_online.order_info
```

PostgreSQL, like other relational database management systems, has a range of additional functions that are useful for generating SQL statements for application development. CURRENT_DATE and CURRENT_TIME are examples of useful functions to explore.

TOTAL_ORDERS

23

It is possible to complete multiple aggregate functions simultaneously. The next example calculates the average, minimum and maximum retail price of all products sold on the Garden Express web store.

```
SELECT AVG(item_price) AS avg_price, MIN(item_price) AS min_price, MAX(item_
price) AS max_price
FROM plants_online.product_details
```

It should be noted that other fields cannot be included in the SELECT statement with aggregate functions unless items are grouped.

AVG_PRICE	MIN_PRICE	MAX_PRICE
8.34	0.99	134.80

Grouping

It is sometimes necessary to group data into subsets. The GROUP BY clause is one of the most useful features of SQL and allows programmers and database administrators to generate powerful SQL statements. The GROUP BY clause is often used in conjunction with aggregate functions. In the examples that follow we will explore the use of the GROUP BY function in this way.

In the previous example, average, minimum and maximum price were extracted from the 'product_details' table using aggregate functions. But in this scenario, more fine-grained results are required. The Sales Manager for Garden Express wishes to know what these values are when broken down by product category. This will help them to determine if more value items should be added to the store. An example solution to this request would be as follows:

```
SELECT item_category_id, AVG(item_price) AS avg_price, MIN(item_price) AS min_price,
       MAX(item_price) AS max_price
FROM plants_online.product_details
GROUP BY item_category_id
```

ITEM_CATEGORY_ID	AVG_PRICE	MIN_PRICE	MAX_PRICE
1	4.34	0.99	8.80
2	3.38	1.23	9.02
3	6.49	2.49	14.50

The results from this query show how the use of the GROUP BY clause has completed calculations for groups of records associated with each item_category_id.

It is also possible to use multiple fields within a GROUP BY clause.

When implementing a GROUP BY clause in a SELECT SQL statement, a general rule of thumb is that, if a non-aggregated field is selected, it should also appear in the GROUP BY clause.



```
SELECT cat_group_desc, cluster_category_desc, COUNT(*) AS clustered_cat_count
FROM plants_online.product_categories
INNER JOIN plants_online.product_cat_group
ON product_categories.cat_group_id = product_cat_group.cat_group_id
GROUP BY cat_group_desc, cluster_category_desc
```

In this example, the subgroup cluster_category_desc within the group cat_group_desc is used to group rows. The aggregate function COUNT is used to count the number of rows in each subgroup. The group and subgroup headings, along with the total count for each subgroup, are included in the result table.

Note: as the 'cat_group_desc' matching field is in a different table (the product_cat_group table), an INNER JOIN is needed to bring necessary fields together.

When you explore the SQL result table, you can see that the grouping has been completed at two levels, firstly by cat_group_desc values, then by the cluster description (including NULLs). The colour coding applied to the table shows the levels of grouping at work.

CAT_GROUP_DESC	CLUSTER_CATEGORY_DESC	CLUSTERED_CAT_COUNT
Tree, Shrubs and Grasses	Foliage Plants	5
Bulbs	Spring Flowering	11
Bulbs	Summer Flowering	10
Edible Produce	NULL	5
Tree, Shrubs and Grasses	Grass and Strap Leaf Plants	9
Tree, Shrubs and Grasses	Flowering Plants	14
Tree, Shrubs and Grasses	Climbing Plants	5
Bulbs	Autumn Flowering	6

An additional feature of the GROUP BY clause is the ability to add conditions against grouped data. The HAVING clause can be used in conjunction with the GROUP BY clause. The HAVING clause is different from the WHERE clause as it attempts to apply conditions to group results.

In the following SQL statement, we will expand on the previous example and filter out any counts that are less than 10.

```
SELECT cat_group_desc, cluster_category_desc, count(*) AS clustered_cat_count
FROM plants_online.product_categories
INNER JOIN plants_online.product_cat_group
ON product_categories.cat_group_id = product_cat_group.cat_group_id
GROUP BY cat_group_desc, cluster_category_desc
HAVING COUNT(*) >= 10
```

CAT_GROUP_DESC	CLUSTER_CATEGORY_DESC	CLUSTERED_CAT_COUNT
Bulbs	Spring Flowering	11
Bulbs	Summer Flowering	10
Tree, Shrubs and Grasses	Flowering Plants	14

In the SQL result table, you can see that the previous record outputs with 5, 9 and 6 have now been excluded from the list.

Subqueries

Although WHERE and HAVING conditions explained earlier are usually used to compare a field value to a single value or pattern, it is also possible to compare the field value to a list of values enclosed in parentheses. The list of values can be supplied as literals by the programmer or dynamically generated using a subquery. Subqueries are SQL SELECT statements used to generate a list of values for use by another query.

To include a subquery within a query, an SQL SELECT statement enclosed within parentheses () is used in the WHERE or HAVING condition.

```
SELECT item_category_id, item_category_desc
FROM plants_online.product_categories
WHERE cat_group_id IN (SELECT cat_group_id
FROM product_cat_group
WHERE cat_group_desc LIKE '%c%')
```

The way to think of the process is that the DBMS will first execute the subquery, producing a list that it will then use in the outer main query.

```
SELECT item_category_id, item_category_desc
FROM plants_online.product_categories
WHERE cat_group_id IN (2,7,8)
```

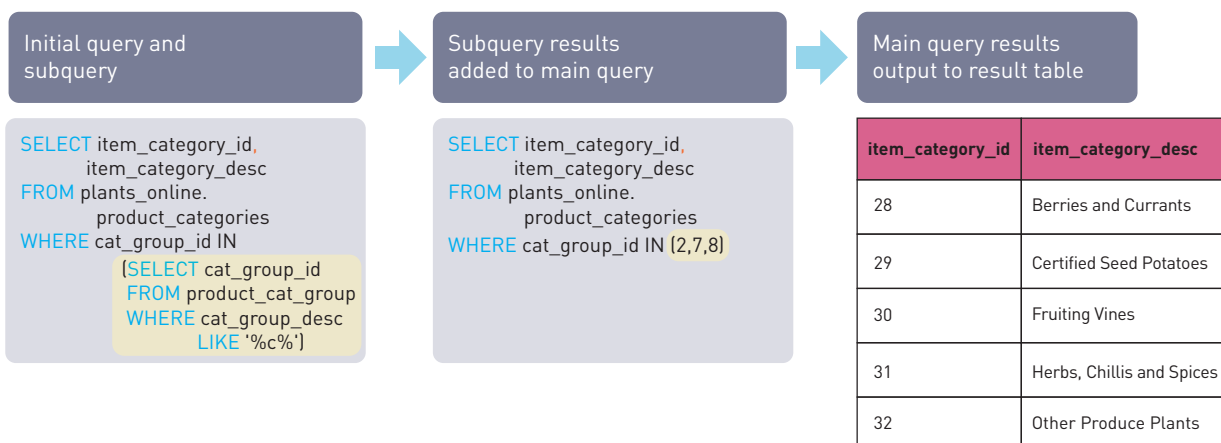


Figure 2.3.12 Process of query and subquery execution

ITEM_CATEGORY_ID	ITEM_CATEGORY_DESC
28	Berries and Currants
29	Certified Seed Potatoes
30	Fruiting Vines
31	Herbs, Chillis and Spices
32	Other Produce Plants

Below are more examples of using subqueries in SQL, and the types of operands available, as well as handy hints about when to apply them.

OPERANDS	EXAMPLE	NOTES
=	This example selects the products from the newest supplier. <pre>SELECT item_id,item_description,item_price FROM plants_online.product_details WHERE supplier_id = (SELECT MAX(supplier_id) FROM supplier_details)</pre>	The = operand can only be successfully used when the results for the subquery itself can only ever be a single result.
IN	This example selects categories where the group name contains the letter 'c'. <pre>SELECT item_category_id, item_category_desc FROM plants_online.product_categories WHERE cat_group_id IN (SELECT cat_group_id FROM product_cat_group WHERE cat_group_desc LIKE '%c%')</pre>	The IN operand is the most versatile and commonly used operand for subqueries. It can be successfully used when one of the results for the subquery match the external field value.
ALL	This example selects the cluster category of the most populated item category. <pre>SELECT cluster_category_desc, COUNT(item_category_desc) FROM plants_online.product_categories GROUP BY cluster_category_desc HAVING COUNT(item_category_desc) >= ALL (SELECT COUNT (item_category_desc) FROM plants_online.product_categories GROUP BY cluster_category_desc)</pre>	The ALL operand is linked with a comparison operator such as < or >. In this example it finds the largest cluster_category group. If <= ALL is used it would find the smallest group.

ORDER BY clause

The ORDER BY clause is the last clause to be added to a SELECT SQL statement. It allows data to be ordered in ascending or descending order and to multiple levels.

The following example of the ORDER BY clause produces a list of customer details and orders the output by last name and then by first name. If no order preference is stated, the list will be in ascending (smallest to largest) order.

```
SELECT customer_id,first_name,last_name
FROM plants_online.customer_details
ORDER BY last_name, first_name
```

The next example orders the list of category descriptions in reverse or descending order. To explicitly reverse the order, the term 'DESC' must be placed after the field in the ORDER BY clause.

```
SELECT item_category_desc
FROM plants_online.product_categories
ORDER BY item_category_desc DESC
```

Subqueries can be nested within each other to multiple levels. It is also possible to compare values between an outer query and a subquery. This is referred to as a correlated subquery.



2.3.14 Nested subqueries and correlated subqueries

INSERT, UPDATE and DELETE statements

The INSERT and DELETE statements are used to add or delete records in data tables, and the UPDATE statement is used to modify existing records. These three statements form the core SQL functionality to modify data contained within the database.

Let's look at some examples of these statements.

INSERT SQL statements

The following example inserts a new category group record into the 'product_cat_groups' table.

```
INSERT INTO plants_online.product_cat_groups (group_cat_id, group_cat_desc)
VALUES (99, 'Garden Tools')
```

The next example inserts a free item against all orders that exist in the database.

```
INSERT INTO plants_online.order_details
(order_id, item_id, qty,
unit_order_price, item_order_notes)
SELECT order_id, 99 AS item_id, 1 AS qty,
'0.0' AS unit_order_price,
'free product' AS item_order_notes
FROM plants_online.order_info
```

DELETE SQL statements

The following example deletes all records from the 'product_cat_groups' table.

```
DELETE FROM plants_online.product_cat_groups
```

This next example deletes a specific record from the 'product_cat_groups' table.

```
DELETE FROM plants_online.product_cat_groups
WHERE group_cat_id = 1
```

UPDATE SQL statements

In this example, the value 'Australia' will be added to the 'country' field against all records.

```
UPDATE plants_online.customer_details
SET country = 'Australia'
```

In this example, the 'email', 'first_name' and 'last_name' fields of customer '1' will be updated with new values.

```
UPDATE plants_online.customer_details
SET email = 'john@abtt.edu.au', first_name = 'John', last_name = 'Smith'
WHERE customer_id = 1
```

In many relational database management systems it is possible to create pre-written, function-like SQL code segments called stored procedures. Stored procedures leverage the power of SQL, and intermix proprietary text-based procedural (or object-oriented) programming syntax to develop robust and abstract data manipulation solutions. While stored procedures will not be explicitly explored in this unit, there are many advantages to deploying them, and good reasons why they are widely used within industry as a best-practice approach. Stored procedures have the useful ability to be called and pass parameters (data). The inner workings of SQL statements as well as the underlying database table structure can then be kept secure and not easily accessible as direct SQL INSERT, UPDATE and DELETE statement calls to the database are not initiated from the application level.



Explore



2.3.15 INSERT
statements
2.3.16 DELETE
statements
2.3.17 UPDATE
statements



Explore

ACTIVITY 2.3.2

- 1 One of the more difficult problems associated with manipulating data from different sources is using date and time information. This is because, in different parts of the world and in legacy systems, dates and times are stored in various formats. In Australia and the United Kingdom, dates are generally formatted as DD/MM/YYYY. In the United States, dates are generally formatted as MM/DD/YYYY. For modern database management systems there are more standardised approaches that alleviate these issues. Your task is to investigate how modern database management systems such as Microsoft SQL Server or MySQL store date and time data.
- 2 Three external datasets relating to weather data for London, New York and Sydney have been sent to you in the form of comma-delimited text files. The three fields within the files relate to the location, the date and the maximum temperature.

London CSV file data examples**'London', '02-12-2011',13.4****'London', '03-12-2011',12.4****'London', '04-12-2011',11.9****New York CSV file data examples****'New York', '12-24-2011',50.6F****'New York', '12-25-2011',44.2F****'New York', '12-26-2011',61.7F****Sydney CSV file data examples****'Sydney', '2011-12-22 13:00:11',33.4****'Sydney', '2011-12-23 12:34:01',29.6****'Sydney', '2011-12-24 14:31:21',28.7**

The data will need to be INSERTED into the database table called 'city_max_temps'. The table fields are called 'location', 'temp_date' and 'max_temp'.

LOCATION	TEMP_DATE	MAX_TEMP
London	2011-01-01	12.2
London	2011-01-02	11.6
New York	2011-01-01	4.5
New York	2011-01-02	3.2
Sydney	2011-01-01	34.4
Sydney	2011-01-02	28.8

Develop an algorithm to insert the data into table records and appropriately mitigate any errors by cleaning the data. Your algorithm should include the SQL query required.



Programming environments with data

As mentioned earlier, different programming environments and relational database management systems use the fundamentals of SQL in slightly different ways. While the choice of programming language and database system will be based on the chosen technology context, in this section we will explore some sample implementations in relation to the Garden Express web technology context scenario.

CASE STUDY



WEB CONTEXT SCENARIO: GENERATED CODED COMPONENTS EXAMPLES

Below are examples of an ASP.NET website application written in the C# programming language. In these examples, data is extracted from the Garden Express database, which is managed by a Microsoft SQL Server database management system. The application then processes this dataset to be presented to the user via the web-based user interface.

Example 1: Displaying a data-driven product list

In this coded example, data is retrieved from the database based on which menu category the user has selected. In this case it is the category related to types of potato crops (categoryID=63). This value is used to complete and trigger the SQL query:

```
string queryString = "SELECT itemid, itemname, imagefile, retailprice FROM
dbo.ProductDetails WHERE category = " +userSelectedCategory+ ";;";
```

If the variable 'userSelectedCategory' is set to 63, the actual SQL statement will be:

```
SELECT itemid, itemname, imagefile, retailprice FROM dbo.ProductDetails WHERE
category = 63;
```

Using automated processes built into the ASP framework, data is then collected in a DataReader code object and attached to the SQL Repeater Object that appears on the screen. This object then continually reads the data saved in the dataReader using the HTML tags and features developed for this to appear as shown. It should be noted that CSS plays a significant part in the layout of the interface in this way. CSS classes such as 'GELoopProduct', 'w-Price-amount', 'w-Price-currencySymbol' and 'add_to_cart' provide these user interface web features.

Products.aspx

This file shows the mixing of HTML tags, CSS class references and ASP.NET server objects (such as Repeater Object) to present data and interact with the user.

```
<ul class="products">

<asp:Repeater ID="ProductRepeater" runat="server">
<ItemTemplate>
<li>
    <a class="GELoopProduct" href="product.aspx?product=<##
    Eval("itemid") %>">
    <img width="160" height="160" class="Geshop_listImage"
    alt="<## Eval("itemname") %>"
    src="product_images/<## Eval("imagefile") %>"
    sizes="(max-width: 160px) 100vw, 160px" >

    <h3><## Eval("itemname") %></ h3>

    <span class="price">
    <ins>
```




```

        <span class="w-Price-amount">

        <span class="w-Price-currencySymbol">$<end span>

        <%#
        Eval("retailprice") %>

        </span>

    <end ins>
</span>
</a>
    <a class="add_to_cart"
    href="cart.aspx?additem=<%#
    Eval("itemid") %>">
Add to Cart<end a>
</li>
</ItemTemplate>
</asp:Repeater>
</ul>

```

Products.aspx.cs

This file shows the C# code written in the page load event. Comments are provided to explain the code using the C# comment syntax //.

```

public partial class sampleproducts2
: System.Web.UI.Page
{
    const string connectionString
    = "Data Source=(local);Initial
    Catalog=GardenExp; User Id = website;
    password=EnterYourPasswordHERE";

    protected void Page_Load(object
    sender, EventArgs e)
    {
        if (!IsPostBack)
        {
            //retrieve category code from page querystring - added when user
            //selected category link on prior page
            string userSelectedCategory = Request.QueryString["category"];
            if (userSelectedCategory == null)
            {
                //no category received - go back to main page
                Server.Transfer("index.aspx");
            }

            //develops a simple query of field from productdetails table where
            //category matches one selected by user
            string queryString = "SELECT itemid, itemname, imagefile,
            retailprice FROM dbo.ProductDetails WHERE category ="
            +userSelectedCategory+ ";";

```

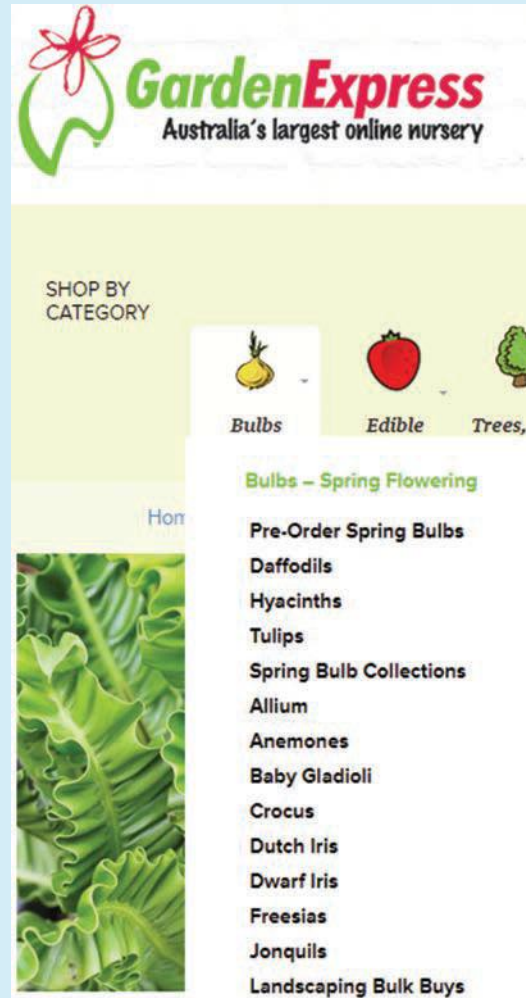


Figure 2.3.13 Products.aspx screenshot

```

using (SqlConnection GEConnection = new
SqlConnection(connectionString))
{
    // Set a new data (SQL) command for the database
    SqlCommand command = new SqlCommand(queryString, GEConnection);

    // Open the Database Connection
    GEConnection.Open();

    // Execute SQL to retrieve data records from database table
    SqlDataReader GardenExpressReader = command.ExecuteReader();

    // Set the DataReader as the repeater objects datasource
    ProductRepeater.DataSource = GardenExpressReader;

    // Bind the data within DataReader to the repeater
    ProductRepeater.DataBind();

    // Close the Database Connection
    GEConnection.Close();
}
}
}
}

```

Example 2: Customer billing address update form

In this example, a customer has logged into their Garden Express account and decided to update their generic billing address details. These details will be applied to orders unless overridden by the customer. The user must be logged in to be allowed to edit this form.

When the form page is loaded, data is 'binded' to the form. This is done by calling the 'bindform' procedure, which achieves this result in a slightly different way to the previous form, but the end result involves data being retrieved from the database and pushed to the data form.

The SQL statement triggered looks something like this:

```

"SELECT
    customerid, billingfirstname, billinglastname, billingcompany,
    billingaddress1, billingaddress2, billingsuburb, billingstate,
    billingpostcode, billingphone, billingemail
FROM
    customers WHERE customerid="+authenticated_user+";"

```

In this case, the 'authenticated_user' variable will contain a specific customer ID value to limit only their data being retrieved. The true SQL statement will look something like this:

```

SELECT
    customerid, billingfirstname, billinglastname, billingcompany,
    billingaddress1, billingaddress2, billingsuburb, billingstate,
    billingpostcode, billingphone, billingemail
FROM
    customers WHERE customerid=14;

```

The user then updates any of the fields as required. If the user has entered information before this, the old information will appear and will still be able to be updated.



When the user selects 'SAVE ADDRESS' at the bottom of the form, it triggers the 'EditBillingInfo' object's 'ItemUpdating' event procedure. This code collects the data from this form. It then constructs a SQL Update statement using the new values the user has entered.

The Update statement in the code looks like this:

```
"UPDATE Customers
SET billingfirstname ='" + txtFirstName.Text + "'," + "billinglastname ='"
+ txtLastName.Text + "'," + "billingcompany ='" + txtCompany.Text + "',"
+ "billingcountry ='" + ddlCountry.SelectedValue + "'," + "billingaddress1
=''" + txtAdd1.Text + "'," + "billingaddress2 ='" + txtAdd1.Text + "'," + +
"billingaddress2 ='" + txtAdd1.Text + "'," "billingsuburb ='" + txtSuburb.
Text + "'," + "billingpostcode ='" + txtPostCode.Text + "'" + "billingphone
=''" + txtPhone.Text + "'," + "billingemail ='" + txtEmail.Text + "'" +
"WHERE CustomerID=" + key.Value.ToString();
```

With sample data, the SQL statement would actually look something like this once it has been successfully concatenated together:

```
UPDATE Customers
SET billingfirstname = 'John', billinglastname = 'Smith', billingcompany = '',
billingcountry = 'Australia', billingaddress1 = '22 New Street', billingaddress2
= '', billingsuburb = 'Maryborough', billingstate = 'Queensland', billingpostcode
= '4650', billingphone = '0741201199', billingemail = 'john@SmithWorld.com'
WHERE CustomerID=12;
```

This Update SQL statement is then executed. Once this is complete, the user is redirected back to the main customer account page [figure 2.3.14].

Billing address *

First name *	Last name *
Company name	
Country *	
Australia	
Street address *	
Suburb *	
State *	
Queensland	
Postcode *	
Phone *	Email address *

SAVE ADDRESS

Figure 2.3.14 Garden Express customer account form

Myaccount.aspx

This file shows the mixing of HTML tags, CSS class references and ASP.NET server objects (e.g. FormView Object) to present data, harvest updated data and interact with the user.

```
<asp:FormView ID="EditBillingInfo" DataKeyNames="customerid"
AllowPaging="false" RunAt="server"
    onitemupdating="EditBillingInfo_ItemUpdating"
    onitemupdated="EditBillingInfo_ItemUpdated" DefaultMode="Edit"
    Width="100%">
    <ItemTemplate>
    <end ItemTemplate>

    <EditItemTemplate>
        <p class="formrowfirst" id="billing_first_name_field">
            First Name <abbr title="required" class="required">*<end abbr>
            <asp:TextBox ID="billing_first_name" Text='<##
            Bind("billingfirstname") %>' RunAt="Server" />
        <end p>
        <p class="formrowlast" id="billing_last_name_field">
            Last Name <abbr title="required" class="required">*<end abbr>
            <asp:TextBox ID="billing_last_name" Text='<##
            Bind("billinglastname") %>' RunAt="Server" />
        <end p>
        <p class="formrowwide" id="billing_company_field">
            Company Name
            <asp:TextBox ID="Company_name" Text='<## Bind("billingcompany") %>'
            RunAt="Server" />
        <end p>
        <p class="formrowwide" id="billing_country_field">
            Country <abbr title="required" class="required">*<end abbr>
            <asp:DropDownList ID="ddlCountry" runat="server" SelectedValue='<##
            Bind("billingcountry") %>'>
                <asp:ListItem Text="Australia" Value="Australia"
                Selected="True"><end asp:ListItem>
                <asp:ListItem Text="New Zealand" Value="New Zealand"><end
                asp:ListItem>
            <end asp:DropDownList>
        <end p>
        <p class="formrowwide" id="billing_address_1_field">
            Street address <abbr title="required" class="required">*<end abbr>
            <asp:TextBox ID="billing_address_1" Text='<##
            Bind("billingaddress1") %>' RunAt="Server" />
        <end p>
        <p class="formrowwide" id="billing_address_2_field">
            &nbsp;
            <asp:TextBox ID="billing_address_2" Text='<##
            Bind("billingaddress2") %>' RunAt="Server" />
        <end p>
        <p class="formrowwide" id="billing_suburb_field">
            Street address <abbr title="required" class="required">*<end abbr>
            <asp:TextBox ID="billing_suburb" Text='<## Bind("billingsuburb") %>'
            RunAt="Server" />
        <end p>
        <p class="formrowwide" id="billing_state_field">
            State <abbr title="required" class="required">*<end abbr>
            <asp:DropDownList ID="dllState" runat="Server" SelectedValue='<##
```

```

Bind("billingstate") %>'>
    <asp:ListItem Text="Australian Capital Territory" Value="ACT"><end
asp:ListItem>
    <asp:ListItem Text="New South Wales" Value="NSW"><end asp:ListItem>
    <asp:ListItem Text="Northern Territory" Value="NT"><end asp:ListItem>
    <asp:ListItem Text="Queensland" Value="QLD"><end asp:ListItem>
    <asp:ListItem Text="South Australia" Value="SA"><end asp:ListItem>
    <asp:ListItem Text="Tasmania" Value="TAS"><end asp:ListItem>
    <asp:ListItem Text="Victoria" Value="VIC" Selected="True"><end
asp:ListItem>
    <asp:ListItem Text="Western Australia" Value="WA"><end asp:ListItem>
<end asp:DropDownList>
<end p>
<p class="formrowwide" id="billing_postcode_field">
    Post Code <abbr title="required" class="required">*<end abbr>
    <asp:TextBox ID="billing_postcode" Text='<%#
Bind("billingpostcode") %>' RunAt="Server" />
<end p>
<p class="formrowwide" id="billing_phone_field">
    Post Code <abbr title="required" class="required">*<end abbr>
    <asp:TextBox ID="billing_phone" Text='<%# Bind("billingphone") %>'
    RunAt="Server" />
<end p>
<p class="formrowwide" id="billing_email_field">
    Post Code <abbr title="required" class="required">*<end abbr>
    <asp:TextBox ID="billing_email" Text='<%# Bind("billingemail") %>'
    RunAt="Server" />
<end p>
<p class="formbutton" id="form_update_button">
<asp:ImageButton ID="UpdateButton" RunAt="Server"
CommandName="Update" class="_1977" ImageAlign="Middle"
ImageUrl="savebutton.png" />
<end p>
<end EditItemTemplate>
<InsertItemTemplate>
<end InsertItemTemplate>
<end asp:FormView>

```

Myaccount.aspx.cs

This file shows the C# code written in the 'page load' event procedure, the 'bind form' procedure and the 'EditBillingInfo' form object's 'ItemUpdating' procedure.

```

public partial class myaccount : System.Web.UI.Page
{
    const string connectionString = "Data Source=(local);Initial
Catalog=GardenExp; User Id = website; Password=D1gItal$S0lut1oNs";
    SqlDataAdapter sqlda = new SqlDataAdapter();
    SqlCommand com = new SqlCommand();
    DataTable dt;

    protected void Page_Load(object sender, EventArgs e)
    {
        if (!IsPostBack)
        {
            bindform();
        }
    }
}

```



```

private void bindform()
{
    string authenticated_user = Context.User.Identity.Name;

    SqlConnection conn = new SqlConnection(connectionString);
    dt = new DataTable();
    com.Connection = conn;
    com.CommandText = "SELECT
customerid,billingfirstname,billinglastname,billingcompany,
billingaddress1,
billingaddress2,billingsuburb,billingstate,billingpostcode,
billingphone,billingemail FROM customers WHERE
customerid="+authenticated_user+"";
    sqlda = new SqlDataAdapter(com);
    sqlda.Fill(dt);
    EditBillingInfo.DataSource = dt;
    EditBillingInfo.DataBind();
}

protected void EditBillingInfo_ItemUpdating(Object sender,
FormViewUpdateEventArgs e)
{
    DataKey key = EditBillingInfo.DataKey;
    TextBox txtFirstName = (TextBox)EditBillingInfo.
FindControl("billing_first_name");
    TextBox txtLastName = (TextBox)EditBillingInfo.
FindControl("billing_last_name");
    TextBox txtCompany = (TextBox)EditBillingInfo.FindControl("billing_
company_name");
    DropDownList ddlCountry = (DropDownList)EditBillingInfo.
FindControl("ddlCountry");
    TextBox txtAdd1 = (TextBox)EditBillingInfo.FindControl("billing_
address_1");
    TextBox txtAdd2 = (TextBox)EditBillingInfo.FindControl("billing_
address_2");
    TextBox txtSuburb = (TextBox)EditBillingInfo.FindControl("billing_
suburb");
    DropDownList ddlState = (DropDownList)EditBillingInfo.
FindControl("dllState");
    TextBox txtPostCode = (TextBox)EditBillingInfo.
FindControl("billing_postcode");
    TextBox txtPhone = (TextBox)EditBillingInfo.FindControl("billing_
phone");
    TextBox txtEmail = (TextBox)EditBillingInfo.FindControl("billing_
email");

    SqlConnection conn = new SqlConnection(connectionString);
    com.Connection = conn;
    com.CommandText = "UPDATE Customers SET billingfirstname ='" +
txtFirstName.Text + "'," +
        "billinglastname ='" + txtLastName.Text + "'," +

```

```

        "billingcompany =" + txtCompany.Text + "," +
        "billingcountry =" + ddlCountry.SelectedValue + "," +
        "billingaddress1 =" + txtAdd1.Text + "," +
        "billingaddress2 =" + txtAdd1.Text + "," +
        "billingsuburb =" + txtSuburb.Text + "," +
        "billingstate =" + ddlState.SelectedValue + "," +
        "billingpostcode =" + txtPostCode.Text + "," +
        "billingphone =" + txtPhone.Text + "," +
        "billingemail =" + txtEmail.Text + " " +
        "WHERE CustomerID=" + key.Value.ToString();

    conn.Open();
    com.ExecuteNonQuery();
    conn.Close();
    Server.Transfer("customeraccounts.aspx", true);
}
}

```

TEST YOUR KNOWLEDGE

Relational schemas

- 1 How are optional fields annotated in a relational schema?
- 2 What is the purpose of the arrowed lines between each part of a relational schema?

Structured Query Language

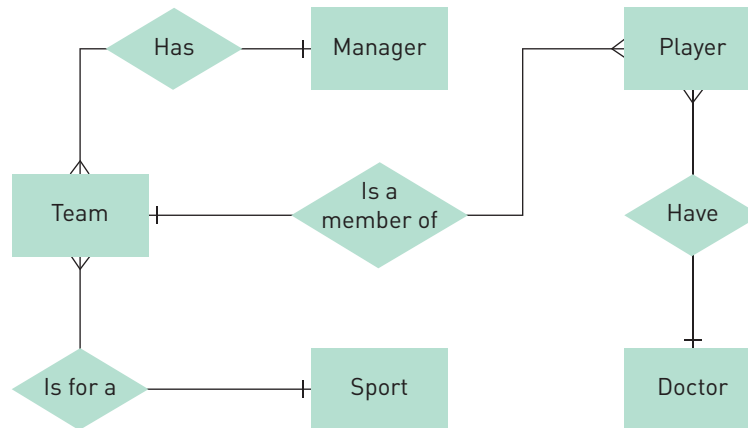
- 1 What is the command for deleting an entire table in SQL?
- 2 Write an SQL statement to add the field 'Age' to the 'Person' table. An example of data is a number between 0 and 100.
- 3 In SQL, what clause is used to help find partial matches to data? Give an example.
- 4 In what situations is dot notation used in SQL statements and why?
- 5 Which logical operator has the *least* importance?
- 6 Produce SQL statements to solve the following problems using the Garden Express database.
 - a Find all product names.
 - b Output all information from the categories table.
 - c Find all products that contain an 'i' and an 'o' within their description (in that order).
 - d Find categories that do not have a Master (Hint: NULL).
 - e The website developer you are assisting requires a list of sale items to be displayed on a specific page for 'Certified Seed Potatoes' (29).
 - f Produce a list of orders for customer 5.



APPLY YOUR KNOWLEDGE

Relational schemas

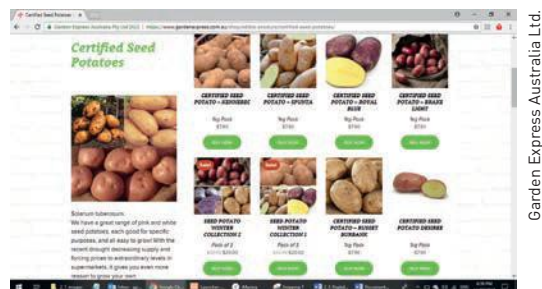
- 1 Use the following ER diagram to construct a data dictionary and a relational schema. Sample example data elements can be added to give meaning.



Structured Query Language

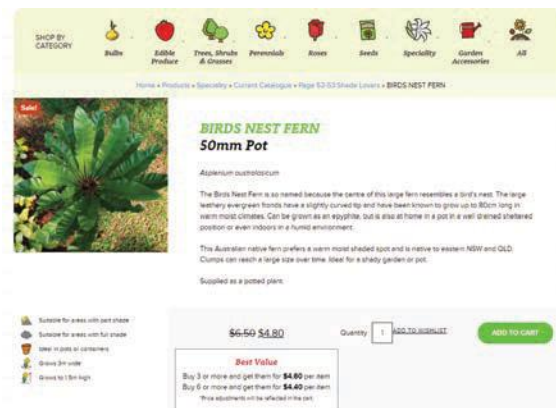
Write SQL statements to solve the following problems using the Garden Express database.

- 2 Produce a list of products that either have an empty master category description or are trees, shrubs or grasses not on sale.
- 3 The following screenshot shows information that has been extracted from the database using two queries. The information has been laid out using visual communication principles and elements. Identify the two distinct information groups, and develop viable SQL queries that could be used for this page.



Garden Express Australia Ltd.

- 4 Review the following screen from the site. Develop an SQL statement to produce all data-driven information for this product page.



Garden Express Australia Ltd.

Data-driven application code

- 8 Using a chosen database management system and programming environment, create a single table database to hold names and address details. Create a test application that connects to the database and allows the user to:

- a display all names and addresses
- b add a new name and address
- c delete an existing name and address.

- 5 Produce a list of payments for order# 5.
- 6 Produce a list of customers who have had multiple orders to different addresses.
- 7 Produce a list of customers who live in the same street and locality.
- 9 Extend the previous activity by filtering the displaying of names and address records by allowing a user to search for partial information (e.g. 'Smi' would display all records containing the characters 'Smi' in the name).

2.4



EVALUATING DATA-DRIVEN APPLICATIONS AND SOLUTIONS

In this section, we will focus on looking at the impacts of data-driven solutions in our society. There will be special attention given to how this affects all of us from both social and economic perspectives. We will explore how data-driven solutions have changed our cultural practices and review if our ethical or legal perspectives have also been altered.

We will also look closely at specific data-driven solutions to monitor their impacts and to work through ways in which generated solutions can be evaluated for their effectiveness in impacting change in developing preferred futures.

Students will evaluate issues and impacts against key concepts relating to the use of data and information in data-driven digital solutions. This includes exploring and evaluating issues and impacts relating to security, integrity, privacy, storage, growth and ethical considerations.

The explosion in data

We live in a world that is creating and consuming data in epic proportions. With continuing growth in personal and commercial use of data, massive data storage centres now drive relentless acquisition and data mining processes that are the outputs of data-driven solutions. The endless creation and delivery of data has driven digital transformation of our society and helped give rise to terms such as 'big data'.

So what has made this explosion of data in the world possible? First, we need to consider the underlying factors to humanity achieving this. We have successfully overcome a number of challenges relating to computing power and cost, the successful development of networks and the Internet across the world and finally making storage cost extremely low and cost-effective. Companies such as Amazon Web Services and Microsoft offer storage and infrastructure as a service, and there are number of other players in the field as well. Big data-driven companies such as Google, Microsoft, Apple and Facebook have built vast arrays of computing and data-storage capacity across the world and are using data as a commodity in their businesses. Many companies consume this data, which is generated by individuals, to offer services like advertising, marketing and other profitable data-driven processes.

GROUP ACTIVITY 2.4.1

In teams of two, investigate data storage as a service online. Look at both end-user focused products (such as Google Drive, Dropbox, OneDrive) and business-oriented paid services such as Amazon Web Services and Microsoft Azure.

List the core features and advantages of these services.

Research the privacy and usage policies regarding the provider's access and protection policies. Summarise positive features and highlight anything of concern.

In class, each group should share one well-researched product and highlight what they found. Class members should then vote on what is the most important feature of storage as a service and what is the biggest concern.

Evolving data security impacts

Data about individuals no longer comes solely from manual input of data collected on forms and is not always protected by locked filing cabinets and password-protected security systems. Mobile and 'Internet of Things' devices provide other opportunities to collect user data such as tracking global positioning system (GPS) coordinates to determine location with the assistance of the 27 GPS satellites in orbit around Earth.

The advantages of enabling location tracking in mobile devices are great. Interactive digital solutions such as turn-by-turn directions, location services, richer social media experiences and photo tagging have become essential tools for many users.

The downside is that organisations can access and use personal location information, to learn more about their existing and future customers.

In the following activity, we look at the real-life issues with the advancement of mobile devices and how new data-driven services are taking advantage of mobile device data sources.

ACTIVITY 2.4.1

Using the article 'Mobile phones and Siri-style assistants a growing threat to online security', answer the following questions.

- 1 What concerns does this story highlight?
- 2 What are some examples of data that a mobile phone creates through use?
- 3 What is 'pattern recognition'?
- 4 What is 'data stitching'? How is it done?
- 5 What privacy implications are there of the use of pattern recognition and data stitching, as described in the article?
- 6 Research 'changes in valuing privacy'. Has there been a cultural shift in this area? Why?
- 7 In teams of two, research methods to protect individuals from identified data threats.



2.4.1 Mobile phones and Siri-style assistants a growing threat to online security

While these innovative approaches to data are intended to improve user experiences and support people in their daily lives, the fallout is that this data can be collected by third parties and used to profile individuals, as a surveillance tool and to gain incredibly detailed information about a person's life.

Information as a commodity

Databases provide significant advantages when managing data. When collected and maintained within a database, sensitive user information can be easily shared by government agencies, financial institutions and health services to support and respond to the growing needs of individuals in our society.

In an Australian context, the Australian Privacy Principles were developed to promote ethical use of an individual's personal information. However, the globalisation of services, the threats of terror-related crime and identity theft and the willingness of end users to share social data have all compounded the potential issues associated with an individual's privacy.

Information can be 'mined' from the Internet just as metal is mined from the ground. Consumer profiles can be traded or sold to other organisations as a valuable commodity. Some organisations actively seek out new information that can be collected, viewed, analysed, mined and (hopefully) disposed of over time. They use these large datasets to gather insights into customer wants and needs, and ultimately innovate their business models to maximise their profits. Consumer loyalty cards are just one way in which Australian businesses are drawing data from their customers for insight on how further marketing can take place.

Supermarket giants Coles and Woolworths use loyalty cards as a means of collecting data. By offering customers a card driven by loyalty points, these organisations are able to flag existing anonymous sales data coming from their registers against individual buys. In effect, customers actively provide the missing link by tying each dataset of purchases to a unique identifier (the loyalty card itself). When a user registers for a loyalty card, they are asked to provide personal information including email address, mobile phone number, date of birth, gender and name.

Title

First Name *

Last Name *

Date of Birth *

Gender * ☐ Female ☐ Male ☐ Not specified

Email *

Confirm Email *

Flybuys will send communications such as statements, special offers, and program news to your email address, subject to the flybuys Privacy Policy.

Mobile

Figure 2.4.1 FlyBuys loyalty card registration page

This information is then matched with the card to provide insights into an individual's shopping tendencies, brand preferences and habits. This is a powerful exercise in **federating data**. The data has the potential to reveal many things about an individual that they may think are private and personal.

ACTIVITY 2.4.2

Think about the issues associated with the collection and federation of data in relation to loyalty card offerings. In the case of supermarket loyalty cards, evaluate the advantages and disadvantages of this data collection process from an individual's perspective.

Information social evolution

For a long time, citizens have been sceptical of governments gaining too much access to personal information and have questioned whether they can be trusted to do so.

We will compare two Australian Government programs that have occurred nearly 30 years apart.



Blackwell, Douglas, www.CartoonStock.com

**"COMPANIES KNOW TOO MUCH ABOUT US, LISTEN...
'YOU'VE EARNED 500 POINTS AND IT'S TIME YOU
GOT BACK IN TOUCH WITH YOUR COUSIN EMMA!'"**

Figure 2.4.2 Data collection and privacy



Figure 2.4.3 The Australia Card was a controversial proposal in the 1980s.

Australia Card, 1980s

In the mid-1980s, the then ALP government of prime minister Bob Hawke embarked on an ambitious campaign and follow-up legislation to introduce the 'Australia Card'. The card was touted as the solution to problems such as tax avoidance and illegal immigration, and as a way to minimise health and welfare fraud committed by citizens. The card was to be a 'super key', a way in which governments had the power to federate all governmental identification systems through interconnection.

The planned process would involve comparing data from more than 20 databases across almost 10 government departments and agencies against the main Medicare register. The Medicare register was thought at that time to be the best and most thorough catalogued identity system for Australian citizens. However, it had been oversubscribed by more than 500 000 individuals, which suggested that it was being used for fraudulent activity within the health system. Each person's identity would be tested between systems to develop an initial federation of data for further refinement and substantiation by each individual. This was to be achieved by making every person in Australia complete an application process that could be followed up with face-to-face meetings to substantiate all information. Individuals could also have been requested to present all verifying

identification documents at this meeting. An individual's signature, and photographs similar to those collected in the application process for an Australian passport, would also be added to the individual's Australia Card before issue.

No one in Australia was to be exempt from this process, regardless of background, age, political, cultural or religious affiliation.

Once cards were allocated to all individuals, a person would be required by law to present that card when gaining employment, seeking government benefits or services, completing banking or financial arrangements, or completing any transactions involving taxation (such as purchasing of property).

The identification card proposal was met with harsh criticisms. Professional groups including the Australian Computer Society made direct submissions to the government, highlighting the need for strong privacy controls and data protections before implementation of a large-scale identification and data system such as the Australia Card. Others in the community compared the identification system to those established by fascist regimes in relatively recent history as a way of controlling and spying on their citizens.

At this point, the Privacy Act was yet to be passed by both houses of Parliament. Many protections are now enshrined in this Act, and guidelines were revised. These are considered a product of those early efforts to change how governments and organisations collected, stored and used data. The Australia Card was in the end abandoned as a policy. However, greater efforts were put into other core systems, such as the Australian Tax File number identification system.

MyGov website, 2010s

MyGov is a centralised authentication portal connecting Australian government online services and is managed by the Department of Human Services. Once an account is created, users link their existing services by supplying critical personal information for challenge-based authentication. New accounts can be created autonomously if the user can prove their identity with specific information, including customer keys, document references and abstract data that only a legitimate user should be aware of.

This challenge-based approach is seen as a safe way to authenticate users' identities and use their existing data to verify their secured access. This is combined with a two-step authentication processes that relies on SMS messages to a mobile phone account nominated by the user.

The system is designed to ease the complication of accessing services online and to amalgamate access in a central system. Within the privacy statement of the myGov website, it is clearly set out what information may be transferred between departments and how this complies with the National Privacy Principles and the Privacy Act.

The privacy statement also alludes to other government policies relating to potential data surveillance. It states that 'the department is required or authorised by law to collect, use or disclose your information for that purpose (including for the purpose of conducting investigations into suspected unlawful activity)'.

In his investigation into government surveillance, journalist Murray Hunter describes how the systems between government agencies are federated to build extremely detailed records about an individual's income, occupation, spouse, children, social security benefits, medical and travel information.

'Data collected by the Australian Taxation Office (ATO), social security (Centrelink), Medicare, immigration, customs, and police enforcement agencies are integrated with relational databases and query systems.'

Murray Hunter

'How Australia is spying on its own', by Murray Hunter, posted 19 November 2013, at www.onlineopinion.com.au/view.asp?article=15692

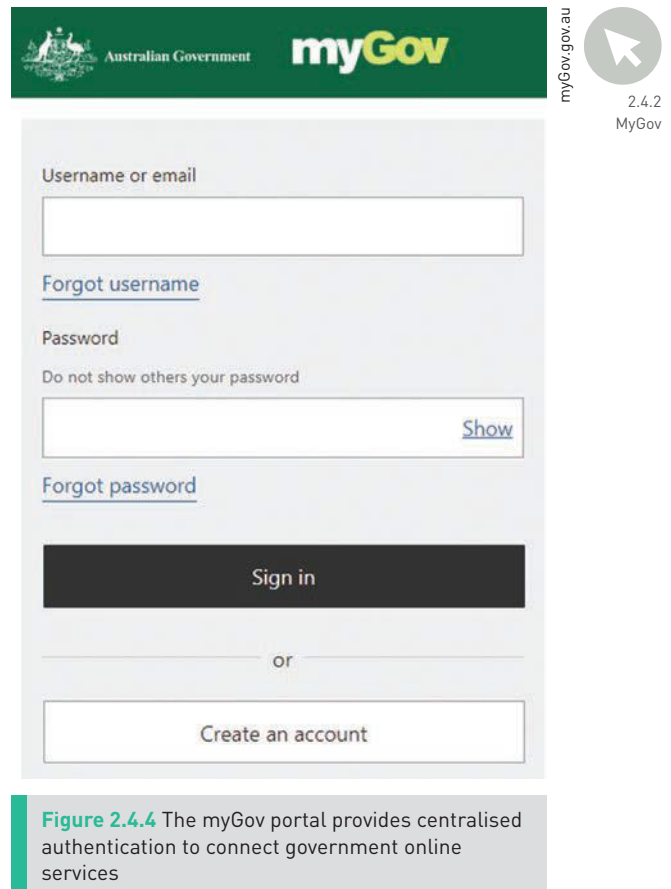


Figure 2.4.4 The myGov portal provides centralised authentication to connect government online services

The verdict

After 30 years, a big question remains: has Australian society matured and evolved to legitimise governmental data identification and surveillance systems, or has the Australia Card program been implemented by stealth over a generation?

Retention and disposal of data

Organisations need to have processes in place that deal with the retention and disposal of data. This is required by the Australian Privacy Act and the Australian Privacy Principles, which specifically state that sensitive personal information that is no longer required needs to be disposed of from recording systems in a secure, timely and reasonable manner. Specifically, Australian Privacy Principle 11, 'Security of personal information', provides guidelines that organisations must follow in destroying or de-identifying personal information. These include the following:

'... in electronic form, reasonable steps will vary depending on the kind of hardware used to store the personal information. In some cases, it may be possible to "sanitise" the hardware to completely remove stored personal information. For hardware that cannot be sanitised, reasonable steps must be taken to destroy the personal information in another way, such as by irretrievably destroying it.'

APP Guidelines, www.oaic.gov.au/agencies-and-organisations/app-guidelines/chapter-11-app-11-security-of-personal-information#destroying-or-de-identifying-personal-information

Getty Images Plus/E+/PeopleImages

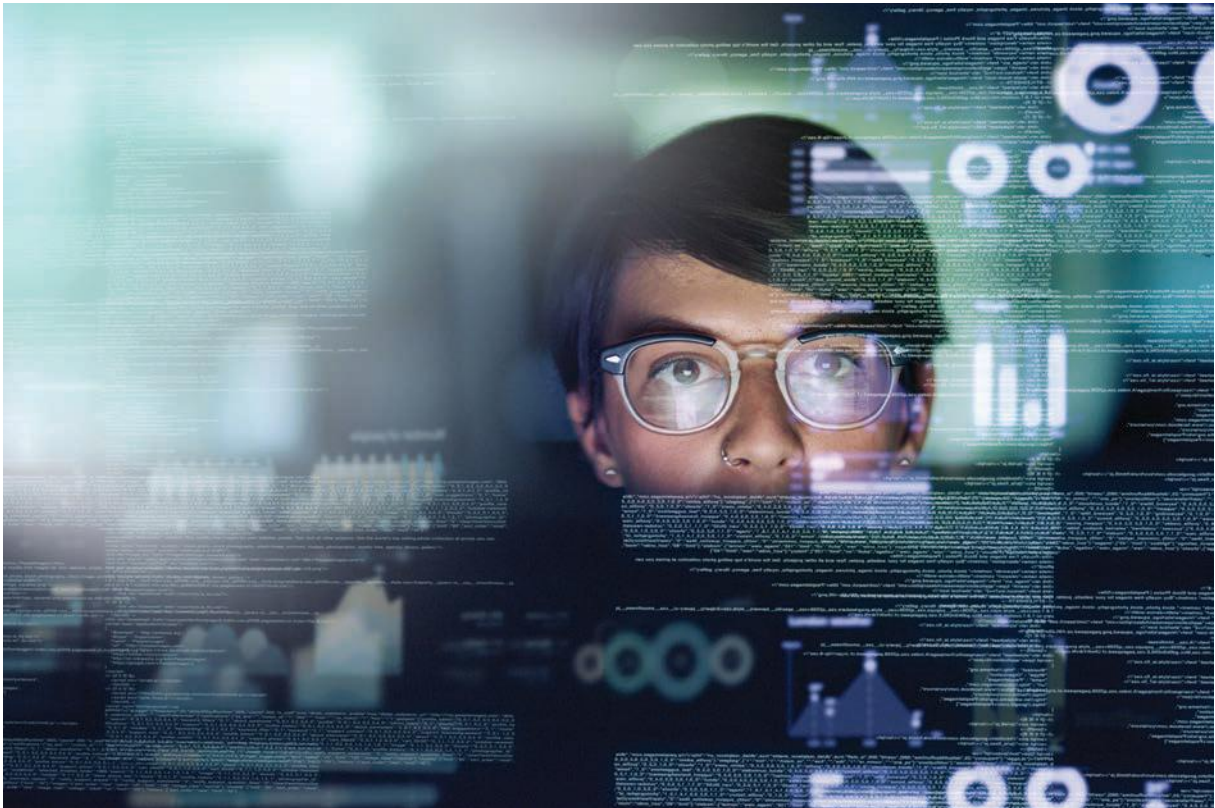


Figure 2.4.5 Data retention laws allow the preservation of metadata for a period of two years.

At other times, companies may be prescribed by law to retain certain types of data for specified times. This is true of the recent government data retention laws that require telecommunication providers and Internet service providers to retain telephony, Internet and email metadata for two years. In this case, **metadata** refers to all the information surrounding the content of communications via voice and data networks, including call times, dates, duration and who was called, as well as IP addresses and email addresses that were communicated with electronically.

As part of these retention laws, criminal law enforcement and intelligence agencies like ASIO and the Australian Federal Police are now able to access this information.

Some in the community feel that these laws cut too deep into the rights of individuals to privacy, and that the government will continue to broaden the scope of how it uses this retained data.

In 2016, the ABC investigated and reported on how government departments were making requests via the Australian Federal Police to investigate metadata information of individuals. The article highlighted the ineffectiveness of safeguards that had been put in place to minimise the release of this metadata.



2.4.3 Data Retention Bill: Government departments ask AFP to access metadata after legislation enacted

De-identification of organisational data

With the rapid growth of data within our institutions, including businesses and governments, people are looking for ways to give external bodies, companies, researchers and the general public ways to use valuable aspects of these datasets without compromising privacy or security. The open data movement encourages institutions around the world to give access to their data for the good of all. The concept is that data that is deemed to be non-sensitive in nature, or can be adjusted so that it is, should be freely available for everyone to use and potentially add value without copyright or licensing restrictions.



Getty Images Plus/E+/AFR

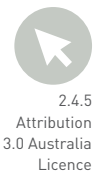
Figure 2.4.6 Open data

In Australia, state and federal governments have declared their support for open data. The data.gov.au website provides a variety of government open data sources that can be accessed as static datasets or via APIs. Information from relational databases is prepared to be published as open data by determining ways to:

- de-identify the dataset for open consumption
- quality-assure de-identification measures through consultations with privacy experts



2.4.4 Explainer: what is the open movement?



- review how the Australian Privacy Principles have been taken into account
- document a data dictionary to help end users to use the datasets.

According to the Australian Privacy Principles guidelines, de-identifying personal information is complete when ‘the information is no longer about an identifiable individual or an individual who is reasonably identifiable’. This process allows departmental heads to approve datasets be released via the data.gov.au website for open consumption under a Creative Commons Attribution 3.0 Australia licence or under proprietary paid licensing arrangements.

Evaluating data-driven solutions

In the past, evaluation has largely been considered something to be done at the end of solution generation. In reality, this does not take into account the iterative nature of development, which requires continual evaluation and refinement throughout the project. The process of exploring, developing, generating and evaluating goals, requirements and choices is essential to evolving and generating the solution.

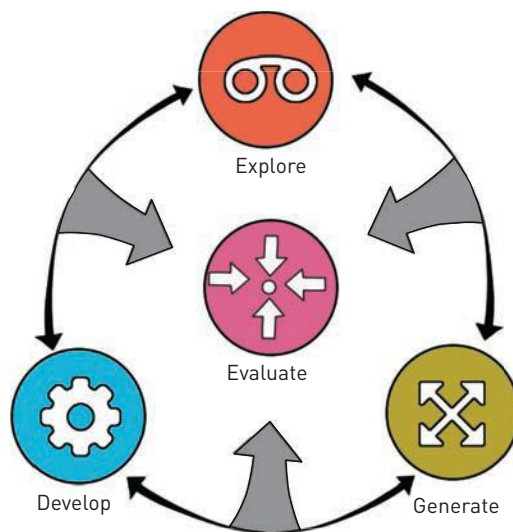


Figure 2.4.7 Evaluation is central to the Digital Solutions problem-solving process.

When you gain a deep understanding of the Digital Solutions problem-solving process, you can see that evaluation occurs everywhere and that, while it plays a significant role at some points, at others it is a smaller but still crucial element in the mix.

In the following sections, evaluation within a data-driven solution context will be highlighted and shown to interrelate to the other phases of the Digital Solutions problem-solving process.

Evaluating while exploring data-driven problems and solutions

Evaluation is critical throughout exploration and investigation. It enables us to gather ideas and information from a range of sources and make judgements about their effectiveness in solving a given problem.

During the investigation of a problem, it is important to ‘mind-map’ and articulate concepts, ideas, issues, requirements and investigations that need to occur. We spend time at this phase informally evaluating the suitability of technology approaches.

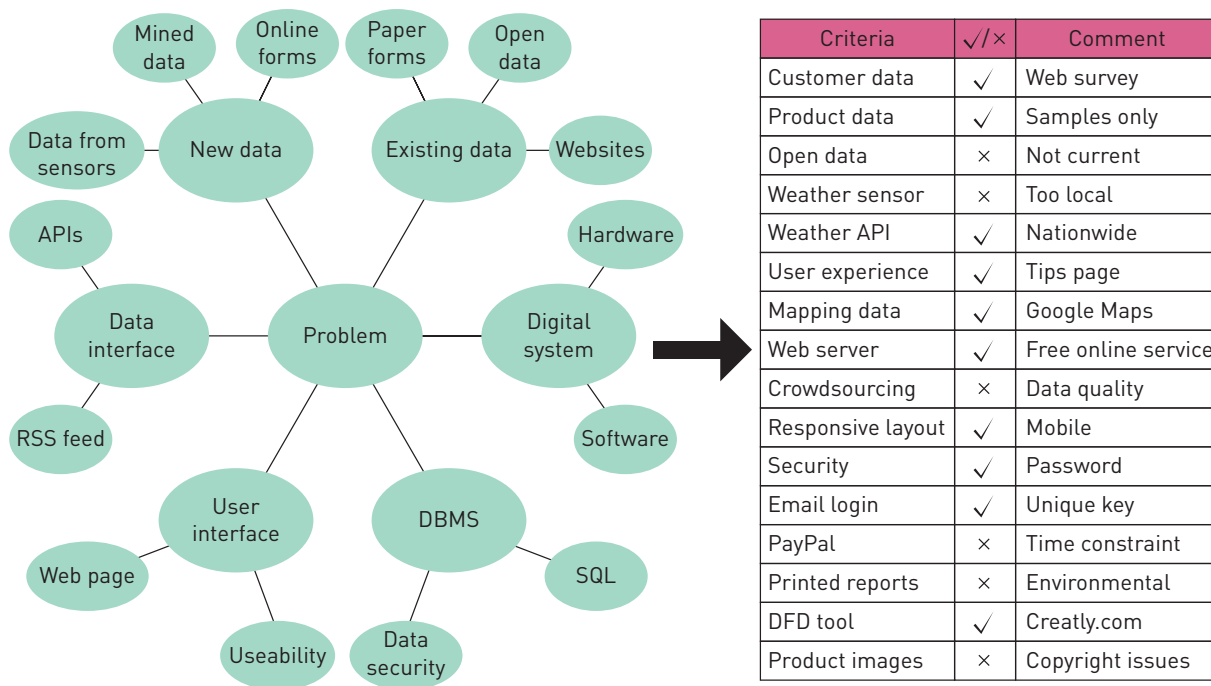


Figure 2.4.8 Mind maps can be used to evaluate the effectiveness of technology processes

We use our mind maps to develop the functional and non-functional requirements for our solution. We gather these from the initial specifications and the exploration of our mind maps, for which we make notes and sketches, and informally document potential solution components. We then form these requirements into criteria to be used as a tool of evaluation that will occur at intervals throughout the solution process. These criteria are continually refined, adjusted and improved as the project owner becomes more informed.

As well as devising these criteria, we evaluate their effectiveness in critiquing the solution:

- Do the criteria effectively describe the requirements of the solution?
- Have the criteria been constructed so as not to be ambiguous to different stakeholders?
- Are the criteria achievable and viable for the project?

Evaluating while developing solutions

During development phases, evaluation takes place when we are presented with different options and justify choices.

When developing the user experience, iterative refinement of ideas results in an improved user interface. Figure 2.4.9 portrays multiple sketches of a mobile application involving **structured data** from a data-driven application. It visualises different approaches, includes informal documented notes to support or repudiate options and ultimately gives indications to developmental judgements for the solution.

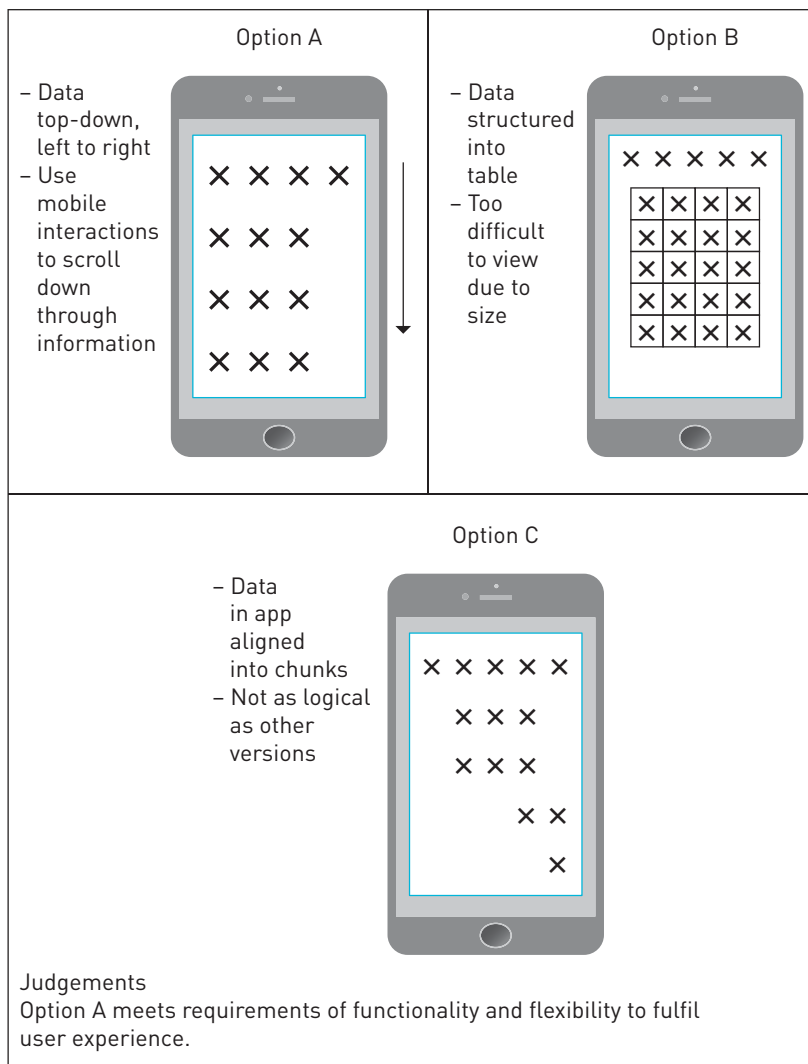


Figure 2.4.9 Evaluating user experience choices

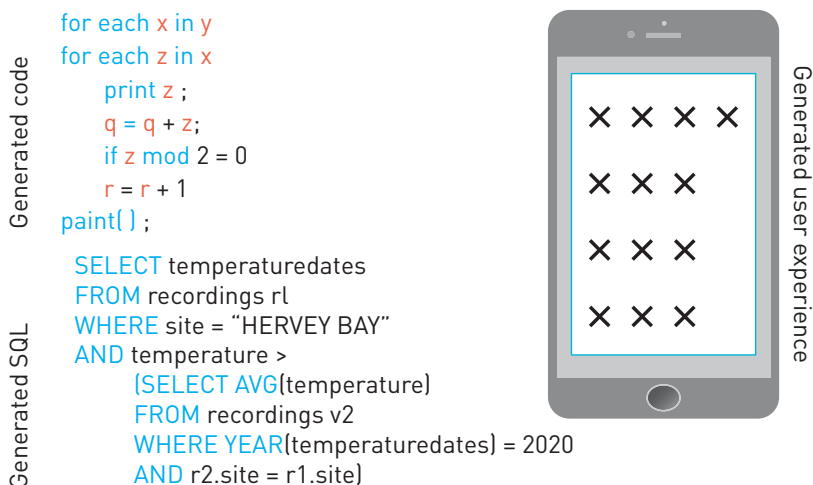


Figure 2.4.10 Evaluating generated components

Evaluating while generating solutions

During the generating phase, solution components are created and assembled. At this phase unit testing is helpful when evaluating the inner workings of components to ensure their effectiveness, robustness and suitability to solve the given problem.

As shown in figure 2.4.10, this may involve comparing written code segments for achieving an outcome; evaluating code segments and judging their suitability for a variety of user scenarios; testing for errors and/or omissions as well as the 'readability' of the code for other programmers who may work with this project one day.

We may also compare our generated user experience with criteria and developmental planning to ascertain its ability to solve the problem, and document recommendations for further refinement beyond the prototyped solution.

Iterative testing of SQL statements may involve initially testing functionality with test data, checking robustness and effectiveness with larger datasets, and possibly load balancing using automated testing processes.

Evaluating solutions

To evaluate a digital solution, we refer back to our refined prescribed and self-determined criteria.

The aim is to determine whether key objectives have been met and to highlight specific issues or details that need further investigation. Criteria checklists allow ongoing evaluation at various determined intervals and, ultimately, at the conclusion of the digital solution. This type of tool helps us to explore the ramifications of meeting, or not meeting, the criteria, to highlight the lessons learned through the problem-solving process.

and to make recommendations for the future. In most cases, the solutions created within the Digital Solutions course of study will be prototypes and will leave room to provide advice on further improvements, needs and innovations.

Functional requirements/Non-functional requirements

Success criteria	Iterations			Details
	1	2	3	
Store customer name and address information	✗			
Generate sales invoices per order		✗		
Stores up-to-date inventory of stocked items			✗	New stock entry screen created post planning
Website adjusts to mobile size device				
Includes client colours as per provided style guide		✗		Adjusted as per evaluation comments
Items are displayed with associated image				
Screen text is easy to read and structured to give clear meaning			✗	
User can update address information				Not completed in prototype as planned
New inventory can be uploaded via CSV file		✗		First attempt caused errors as data was not cleaned

Figure 2.4.11 Evaluating with criteria

TEST YOUR KNOWLEDGE

- 1 What examples of data are generated from a smartphone?
- 2 What is the data purpose behind brand loyalty cards?
- 3 What is the term given to linking data between different databases?
- 4 What are the Australian Privacy Principles?
- 5 Explain the concept of data mining. Give one reason why companies such as Facebook and Google do this.
- 6 Why must companies have a process for destroying data?
- 7 What is data retention?
- 8 What is metadata? Give two examples.
- 9 Describe the positives and negatives of Australia's data retention laws.
- 10 When repurposed and shared beyond its declared use, why must data be de-identified?
- 11 What are the four actions that are recommended when de-identifying data for open use?
- 12 What is the purpose of evaluation? Why is it done?
- 13 How does evaluation occur in relation to developing a data-driven solution? Give an example.
- 14 What is the difference between prescribed and self-determined criteria?



2.4.6 Open data and the G20: the value is there to share
2.4.7 Australian Government public data

APPLY YOUR KNOWLEDGE

- 1 Research Australia's controversial data retention laws. What safeguards have been put in place by government to ensure privacy of the individual? Evaluate your findings and make decisions about the effectiveness of these laws and the safeguards in relation to:
 - a the individual
 - b Australian society
 - c the Australian economy.
- 2 Review the article found at the weblink.
 - a Using the data.gov.au site, find an example of de-identified data. Unpack the resource and list what information has been removed.
 - b List ways in which this data could be used in a data-driven solution.

Research in action

- 3 In groups of four, devise a survey using an online surveying tool that asks users to respond to up to 10 questions about how

governments should be able to gain, store, archive and access an individual's personal data history. Provide example scenarios to gauge respondents' answers about whether it is acceptable or not. Try to ask closed questions that allow respondents to give binary answers (Yes/No) or multiple-choice responses (from 1 for 'I strongly disagree' to 5 for 'I strongly agree').

Present the survey finding to the class for discussion.

Write a class summary of insights into the surveyed data from all groups.

- 4 Using the Garden Express web technology context example in Unit 2, use the solutions requirements listed in section 1 to develop a checklist of criteria.

Unit

3

DIGITAL INNOVATION

INTRODUCTION

Digital innovation occurs when we harness the power of computing to improve our world. It allows us to create early-warning systems based on data collected from electronic sensors. It is evident in mobile apps that allow families in remote areas to access life-saving specialist information. It allows wheelchair-bound students in Australia to experience the view from the top of Mt Everest. It allows an environmentally focused primary school student to prevent deforestation by monitoring online data using a website.

By the end of this unit you will understand how to use the problem-solving process to analyse a problem and generate an innovative prototype of a digital solution.

You will use systems thinking to explore a number of case studies to learn how to recognise and describe the different parts of a digital solution and how to symbolise and explain how the different parts work together. You will explore concepts related to innovative digital solutions and identify areas for further investigation and research.

You will use design thinking to explore innovative ideas for problems that are relevant to you, and analyse problems to determine the user, programming and data requirements of possible solutions. You will design the individual parts of possible solutions and use diagrams to show how the parts will work together to solve the problem.

You will use computational thinking to synthesise information and ideas and create programmed components, user interfaces and databases. You will establish a development environment that allows you to demonstrate how the different components of your prototype solution interconnect to solve a problem. You will run simulations and collect data about the accuracy and efficiency of solution components.

You will analyse test data and user experiences to evaluate the quality and completeness of the solution and make recommendations based on specific criteria.



Explore



Develop



Generate



Evaluate

KEY IDEAS

- Programming elements
- Digital system
- User interface
- Useability principles
- Programming information
- Data structures
- User experiences
- Problems
- Technology context
- Solution requirements
- Prescribed criteria
- Self-determined criteria
- Data elements
- Programmed components
- Prototype digital solution
- Impacts
- Refinements



3.1

EXPLORING INNOVATION



Explore

In this section you will use systems thinking to explore a number of case studies to learn how to recognise and describe the different parts of a digital solution and how to symbolise and explain how the different parts work together. You will explore concepts related to innovative digital solutions and identify areas for further investigation and research.



Explore



3.1.1 Koala Tracker

CASE STUDY

KOALA TRACKER: WEB APPLICATION CASE STUDY

To understand innovative solutions within the web application technology context, the following pages explore a case study of an existing web application to recognise the elements of the system and explain the interactions between each component.

The Koala Tracker application was created in 2009 to solve a problem: 'the body of data on koala populations is patchy, often sparse and not nationally comprehensive or coordinated'. Conservationist Alex Harris wanted to use **crowdsourcing** to collect data about the numbers of koalas living in the wild in order to determine whether or not the koala should be registered as an endangered species. Crowdsourcing is the process of collecting data from a large number of volunteers using the Internet. To facilitate this, Alex designed a web application with the purpose of collecting information on community koala sightings to answer the following three questions:

- 1 How many koalas are left?
- 2 Where do they live?
- 3 What condition are they in?

The **prescribed requirements** of her system were that it should:

- provide an easy-to-use web interface that allows community members, using their phone or laptop, to manually enter koala data about:
 - a location
 - b points of impact
 - c causes of death and injury
- allow database searches on town or suburb, date range, cause of death or injury or keyword
- present the data visually on an interactive map of Australia using green markers for live koalas, yellow markers for sick or injured koalas and red markers to show points where koalas have died
- allow free downloads of the data for use by governments, councils, conservationists and general community members.

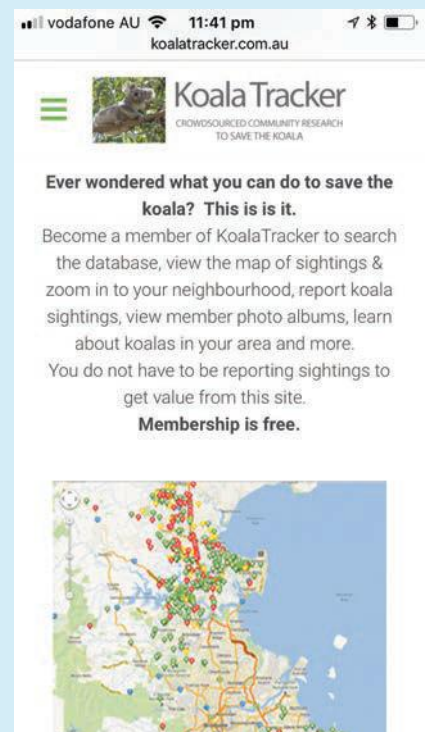


Figure 3.1.1 Screenshot of the Koala Tracker app as it appears on a mobile device

Alex Harris, Creator of KoalaTracker.com.au

The current web application provides easy-to-use web forms that meet the first two prescribed criteria. Early versions of the web application also provided a publicly accessible interactive map and data download feature, but the ongoing monthly charges required to maintain the online mapping feature made it economically unsustainable, so this feature has been removed until an open source solution is available.

The impact of the Koala Tracker web application is that:

- it provides evidence of koala populations in areas of proposed development, which allows governments to block developments that have the potential to decimate colonies of koalas
- it involves the community in the collection of data about koalas
- it generates community awareness and education that leads to local conservation activities.

Although login details are required to access the application, only a name and email address are required to set up a login, and other personal information is optional. If members choose to add additional personal details, these details are protected with password security.

Because the data is crowdsourced, there is no way of verifying the quality of the data. There is also no feature that allows users to update details of their sightings, but this is planned for a future version.

Elements of a web application

Apart from the Internet, the essential elements of a simple web application are a user interface and a server interface. Many web applications such as Koala Tracker also include a data interface. The Koala Tracker web application uses a three-tier architecture, which is illustrated in figure 3.1.2.

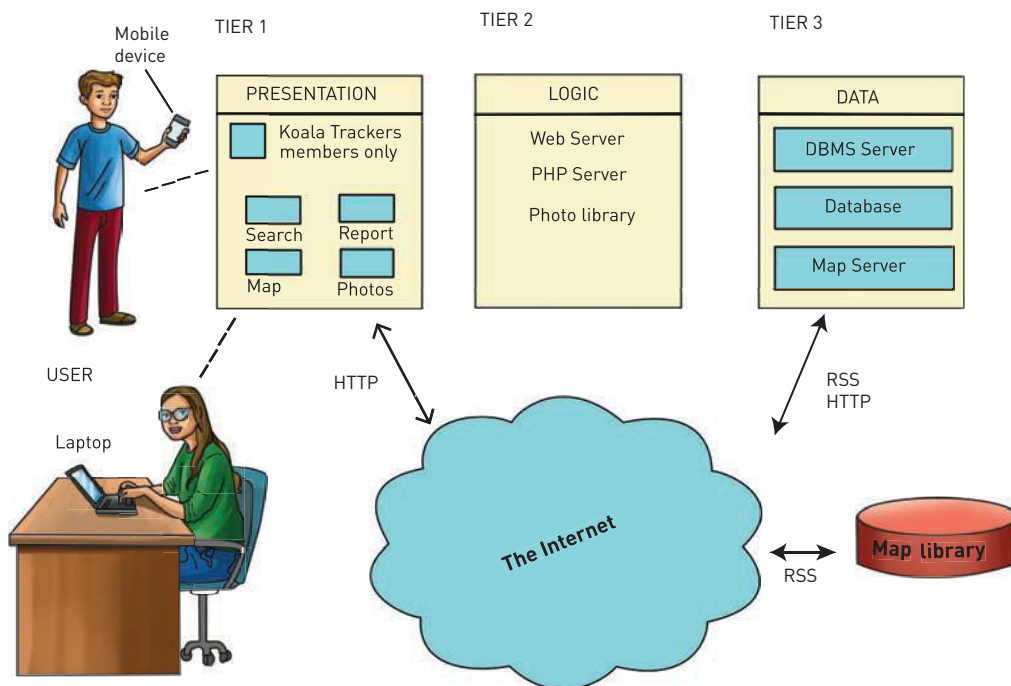


Figure 3.1.2 Elements of the Koala Tracker web application

These tiers work together to process data and present it to the user as information.

Tier 1 Presentation components include the web browser and user devices (user interface).

The web browser processes HTML to mark up content, CSS for styling the content and JavaScript code to add functionality such as user interactions to the content.

Tier 2 Logic components include the web server, PHP server and local storage (server interface).

User input is sent via the Internet to the logic layer, which is a web server hosted by an Internet service provider. The web server manages multiple interactions from many different users at the same time. It provides file storage for uploaded images and manages interactions with both users and other data-processing servers, such as by sending PHP code to be processed by a PHP server and SQL code to be processed by a database management system (DBMS).

Tier 3 Data components include the DBMS server, database and external storage (data interface).

The web server sends data update and retrieval requests to the DBMS server for processing. The DBMS server component of a web application can be stored at the same location as the web server or accessed via the Internet at another location. In the case of the Koala Tracker app, some of the data is stored on the local web server. The mapping data is delivered to the application as a live data feed using the RSS protocol. Unit 3, 'Web applications', will focus on interaction with a local database but live data feeds will be explored further in Unit 4.

PHP is the most popular server-side scripting language for web applications and is the basis for the server logic in Facebook, Wikipedia and many other popular web applications. When a web server recognises PHP code, the entire code segment is sent to a PHP server for processing. The PHP server processes the PHP script and sends the resulting output back to the web server to replace the part of the web page that originally contained PHP code. This way the web page can dynamically update itself by populating pages with data relevant to user selections.

PHP has been used to populate web pages with external data for the last two decades and PHP libraries are regularly reviewed and revised to provide new improved features. However, PHP is not always the best server-side scripting language to use. Other web development frameworks that can manage the logic requirements of the web server include Ruby on Rails, Python and Node.js. Before deciding on a particular approach, it is important to evaluate the alternatives.

RUBY ON RAILS	- Ruby on Rails is well known for its extensive libraries that streamline development, but the same features that make it fast to build apps add additional processing overheads. - Twitter initially used Ruby on Rails but had to switch some server components to a Java server so Twitter could handle the large volumes of tweets that occur during international events.
PYTHON	- YouTube and Pinterest use Python code to manage their web application logic. - Python is considered one of the more efficient languages for server-side processing and is often easier to maintain thanks to its simplicity.
NODE.JS	- LinkedIn and Paypal use Node.js because it processes real-time data efficiently. - It is a new server-side language that has become popular for web applications that stream live data or have a chatroom feature.

User interactions

Digital solutions need to provide a positive user experience. For a web application, this means that the web page the user will interact with is presented in a pleasing way that is easy to use.

The Koala Tracker user interface is responsive so that it can be used on a small phone screen or a larger computer screen.

An interactive map allows community members to easily map latitude and longitude geocodes using a street address to zoom in to a specific area of the map and then drag the marker to the exact area of the sighting. Contextual instructions are provided to improve the useability of the application.

The data form uses drop-down lists, date picker widgets and radio buttons, improving useability and reducing the need for data validation of manual user inputs.

All images this page courtesy Alex Harris, Creator of KoalaTracker.com.au

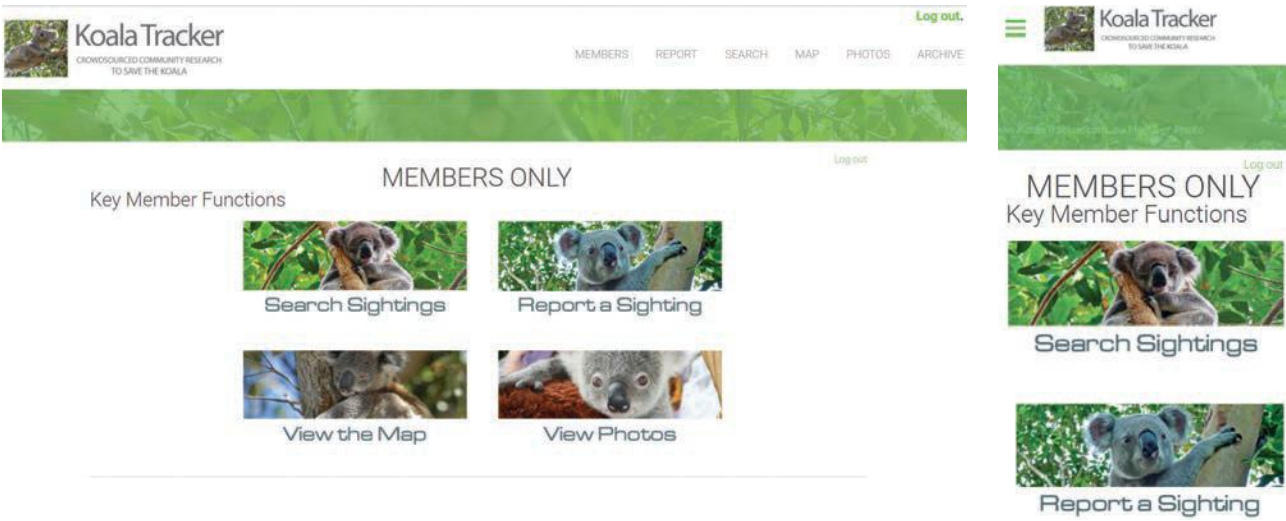


Figure 3.1.3 The main menu of the Koala Tracker web application has a responsive design. The layout changes depending on whether it is viewed on a computer screen (left) or a mobile device (right).

Start Here

1. Type nearest address or landmark, select from dropdown

Brisbane, Queensland, Australia

2. Drag the marker on map to location of koala

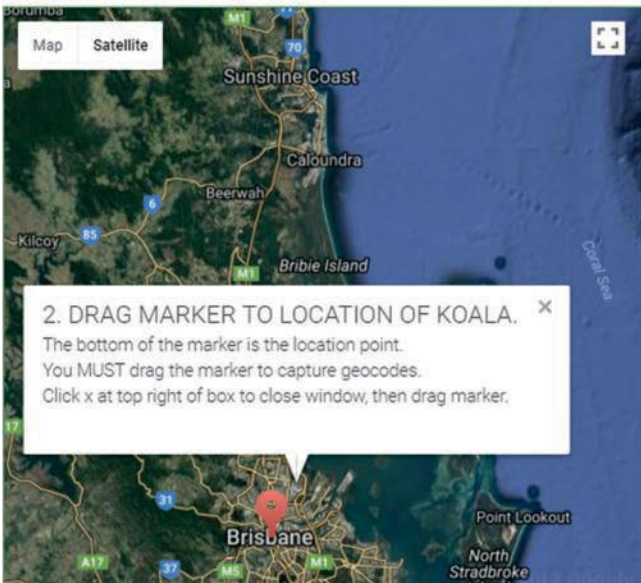


Figure 3.1.4 When a location is entered, the map zooms to a specific location and provides an instruction box, prompting users to drag the marker to the location of their koala sighting.

Select City/Town/Suburb* Start typing and select name when it shows. On tablet or phone, scroll to suburb name. Suburbs are in alpha order.

Brisbane

How many koalas did you see? (at that location at that time) *

1

What date was koala/s seen? *

Jan 2018						
Su	Mo	Tu	We	Th	Fr	Sa
	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	31			

Today Clear Close

- ☐ injured-by fire
- ☐ Injured-unknown
- ☐ Sick-by disease
- ☐ Dead-by car
- ☐ Dead-by cow
- ☐ Dead-by dog
- ☐ Dead-by disease
- ☐ Dead-by fire
- ☐ Dead-by snake
- ☐ Dead-from treefall
- ☐ Dead-unknown

Have you seen koalas here before? *

- ☐ Yes
- ☐ No

Figure 3.1.5 Reducing user options to allow only valid input options improves the useability of the data interface and the quality of the data collection.

Data in the Koala Tracker web application is displayed as the results of a search using an HTML/CSS formatted table. In an earlier version of the app, it was also possible to display red, yellow and green markers for dead, injured and living koalas on an interactive map.

Data processes and system interactions

The koala database has a very simple schema:

Koalas [ID, State, Location, SightingDate, NumberOfKoalas, KoalasSeenBeforeAtThisLocation, Latitude, Longitude, Situation, PhotoFilename1, PhotoFilename2, PhotoFilename 3].

The data dictionary for the koala database is shown in table 3.1.1.

Table 3.1.1 Data dictionary for the koala database

DATA ITEM	DATA TYPE	VALIDATION RULES	EXAMPLE DATA
ID	INTEGER	Primary Key	3492917
State	VARCHAR(3)	Select from list	QLD
Location	VARCHAR(64)		Noosa Heads
SightingDate	DATE	DD-MMM-YYYY	09-Dec-2017
NumberOfKoalas	INTEGER	0 < x < 9	3
KoalasSeenBeforeAtThisLocation	BOOLEAN*	Yes or No	No
Status	VARCHAR(32)	Select from list	Alive
Latitude	DECIMAL	Should match location	-26.410275564005474
Longitude	DECIMAL	Should match location	153.07987930852892
Situation	TEXT**		3 koalas spotted and photographed by GL. One in a tree in front of her house and 2 other koalas in a big blue gum tree at the end of the drive, either fighting or mating. In one of the photos one koala seems to be holding a joey in its arms.
PhotoFilename1	VARCHAR(128)		3492917_95179_25181806_10155926584074231_574033759_n.jpg
PhotoFilename2	VARCHAR(128)		3492917_97316_25188210_10155926584139231_1859886166_n.jpg
PhotoFilename3	VARCHAR(128)		3492917_97317_25188210_10155926584139231_1859886166_n%20-%20Version%202.jpg

* The BOOLEAN data type is supported by most DBMS servers but is not standard ANSI SQL. A BIT or INTEGER value is sometimes used instead of BOOLEAN.

** The TEXT data type is supported by many DBMS servers but is not standard ANSI SQL. VARCHAR(255) can be used in place of TEXT if required.

Data flow diagrams (DFDs) can be used to show the flow of data through the Koala Tracker application. A level 1 data flow diagram is shown in figure 3.1.6.

Control mechanisms

The Koala Tracker web application provides an easy-to-use web interface to update and access data in the koala database, which is stored at a different location on a different server. An HTML table structure is used to present the data in a web page, as shown in figure 3.1.7. The cascading style sheet named `main_style.css` is used to add spacing, alignment and colour to style the content. For example, the green colour to the clickable link in figure 3.1.7 is applied using the following CSS code in `main_style.css`.

```
a { color: #71bd52; }
```

The CSS class named `wsite-multicol-col` is applied to each table cell using the following CSS code:

```
.wsite-multicol-col {
    max-width: 100% !important;
    display: block;
    width: auto !important;
    margin: 0 auto 1em !important;
}
```

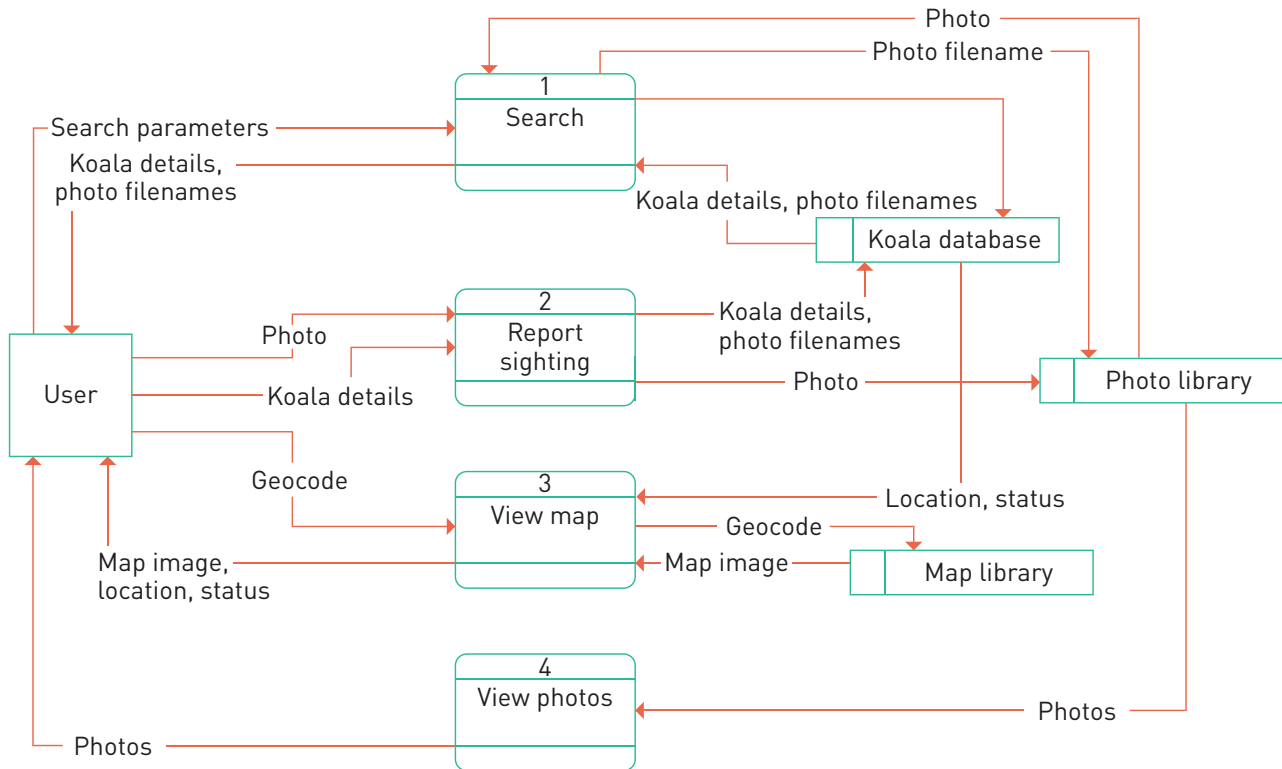



Figure 3.1.6 The four functions available on the main menu are represented as processes in the level 1 data flow diagram.

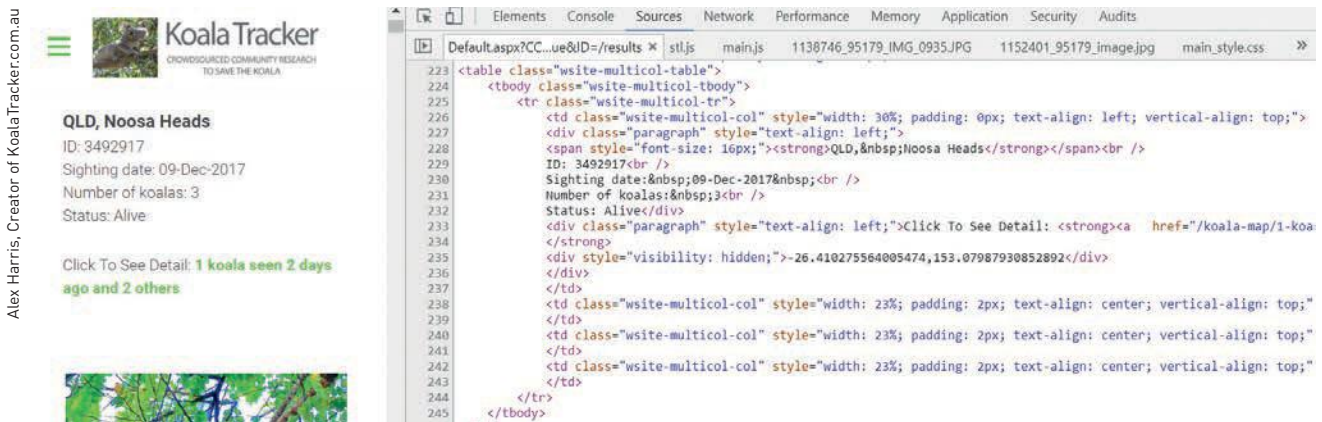


Figure 3.1.7 Using code view in a browser allows you to see the code structures used to create a web page.

The use of **important** is common in user stylesheets to override styles in an underlying template. At the time the Koala Tracker website was created, this was a common quick fix, but it is not considered good programming practice today as it overrides the natural cascading properties of styles in a stylesheet. Web standards organisations continue to revise and improve web programming languages and guidelines, but many older websites continue to use older methods. The **!important** tag, while not recommended for extensive use, is still valid CSS. It is always a good idea to use an online validation tool to check web application code before publishing a web page.





Explore

3.1.3 CSS
validation
service

ACTIVITY 3.1.1

Use an online CSS validator to validate CSS code from the Koala Tracker website.

- 1 Visit the CSS Validation Service.
- 2 In the input window type the CSS code for the Koala Tracker data display class:

```
.wsite-multicol-col {
    max-width: 100% !important;
    display: block;
    width: auto !important;
    margin: 0 auto 1em !important;
}
```

- 3 Click the check button to see if the code is validated without errors.
- 4 Select the By URI tab at the top of the web page.
- 5 Enter your school website address into the Address field and check whether any validation errors are detected.
- 6 Reflect on how this tool could be used to test and evaluate your own CSS code in future projects.

When collecting user input it is important to check the data complies with the validation rules defined in the data dictionary. The Koala Tracker web application validates form fields using JavaScript functions before submitting form data. One of the validation checks used is whether or not data has been entered in all the required fields. The following JavaScript code is used to check that values have been entered in each field:

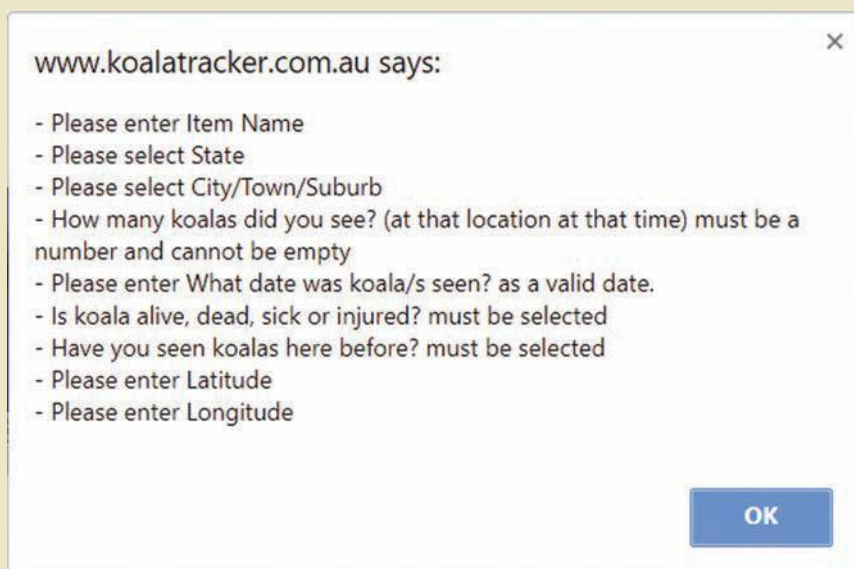
```
function isEmpty(strng, FieldName) {
    var error = "";
    if (strng.trim().length == 0) {
        error = validateLang.Enter.PleaseEnter + FieldName + "\n";
    }
    return error;
}
```

ACTIVITY 3.1.2



Develop

The pop-up shown in figure 3.1.8 is displayed when you attempt to submit the Koala Tracker form without entering a koala sighting.



Alex Harris, Creator of KoalaTracker.com.au

Figure 3.1.8 Error feedback for the Koala Tracker web application

The validation algorithms used to check the form are listed in table 3.1.2. Which of the error messages displayed above have been generated by the isEmpty algorithm?

Table 3.1.2 Validation algorithms used to check the Koala Tracker form

FUNCTION NAME	ISEMPTY	ISNUMERIC	CHECKSELECTED
Parameters	<i>input, fieldName</i>	<i>input, errorText</i>	<i>input, fieldName</i>
Returns	<i>error</i>	<i>error</i>	<i>error</i>
Algorithm	SET <i>error</i> TO "" Remove spaces from <i>input</i> IF length of <i>input</i> = 0 THEN SET <i>error</i> to "Please enter " ADD <i>fieldName</i> to <i>error</i> ENDIF OUTPUT <i>error</i>	SET <i>error</i> TO "" IF length of <i>input</i> = 0 THEN SET <i>error</i> to <i>errorText</i> ELSE REPEAT GET next letter FROM <i>input</i> IF letter is not 0-9 THEN SET <i>error</i> TO <i>errorText</i> ENDIF UNTIL no more letters ENDIF OUTPUT <i>error</i>	IF <i>input</i> is checked THEN SET <i>error</i> TO "" ELSE IF field type is drop down THEN SET <i>error</i> to "Please select " ADD <i>fieldName</i> to <i>error</i> ELSE SET <i>error</i> to <i>fieldName</i> ADD "must be selected" to <i>fieldName</i> ENDIF ENDIF OUTPUT <i>error</i>

CASE STUDY

MOBILE APPLICATION AND INTERACTIVE MEDIA CASE STUDY

To understand innovative interactive media solutions within the mobile application technology context, the following pages explore a case study of an existing mobile game to recognise the elements of the system and explain the interactions between each component.

Fruit Ninja

Fruit Ninja was developed in 2010 by Brisbane-based game studio Halfbrick Studios with one major focus – to create a popular original game that took advantage of user interactions unique to mobile phones. Halfbrick Studios' initial research showed that the most successful mobile games were very simple, used a single screen and took advantage of the touchscreen, and the constraints for its new indie game were built



Figure 3.1.9 The initial pitch for Fruit Ninja received a mixed response from the company. Some developers loved it, but most of the decision-makers thought the idea was too simple to work.



Explore



3.1.4 Fruit
Ninja

Halfbrick Studios Pty Ltd

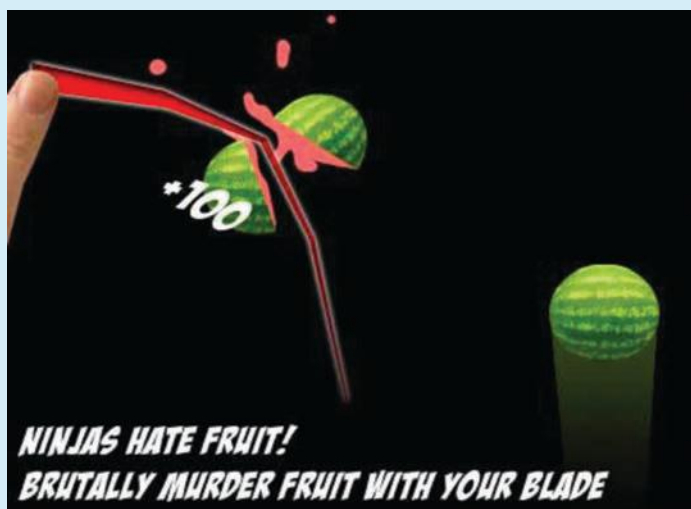


Figure 3.1.10 The first artwork developed for the Fruit Ninja mobile application



Figure 3.1.11 The opening scene of the official Fruit Ninja application, released on iPhone in 2010

on this research. It was decided to constrain developers to focus on the touch interface as a central feature and limit the complexity to 'a single-screen mobile game that your mum could play'. Initial brainstorming using design thinking, and a late-night infomercial about kitchen knives, led to the idea of using finger swipes to slice fruit. The lead designer, Luke Muscat, pitched the idea to other developers at the company using some early artwork scenes shown in figure 3.1.9.

More than one billion people have downloaded and played Fruit Ninja since its initial release, and the game has evolved with each release. The first artwork developed for Fruit Ninja (see figure 3.1.10) was for a simple game interface with one background, one blade, one game mode and no high-score feature.

After four months of iterative development using a scrum methodology, the opening screen had been refined into the screenshot shown in figure 3.1.11, and the game was ready to release to the open market.

The Fruit Ninja game met the criteria prescribed by Halfbrick's CEO, but developers did not consider it a success until it met their self-determined criteria. They were hoping

the game would be downloaded 100 000 times to generate some much-needed revenue that would allow them to keep their jobs. When they first checked this statistic, they realised that they already had over one million downloads. It was then that they knew the game was a success.

After a successful launch on iPhone, Halfbrick Studios developed versions for other mobile phone platforms and now has a large team of developers dedicated to ongoing improvements and maintenance of the Fruit Ninja game. The current version of Fruit Ninja has multiple game modes, blade options, characters and numerous customisable features that were not imagined in the first release. There is even an educational version that teaches maths skills.

In addition to mobile phone platforms, Halfbrick also ported the game to Xbox Kinect, and found that the motion sensor interface was just as innovative on Xbox as the finger swipe game mechanic on the mobile phone platform.

Elements of mobile applications and interactive media

Mobile games are part of both the mobile application and the interactive media technology contexts. The elements of a mobile application prototype are usually confined to the internal hardware, software and data components of the mobile device. More advanced mobile applications usually require interaction with external application servers and shared data as well. Mobile networks connect to the Internet via Internet service providers (ISPs). The components of a typical mobile game, such as Fruit Ninja, are shown in figure 3.1.12.

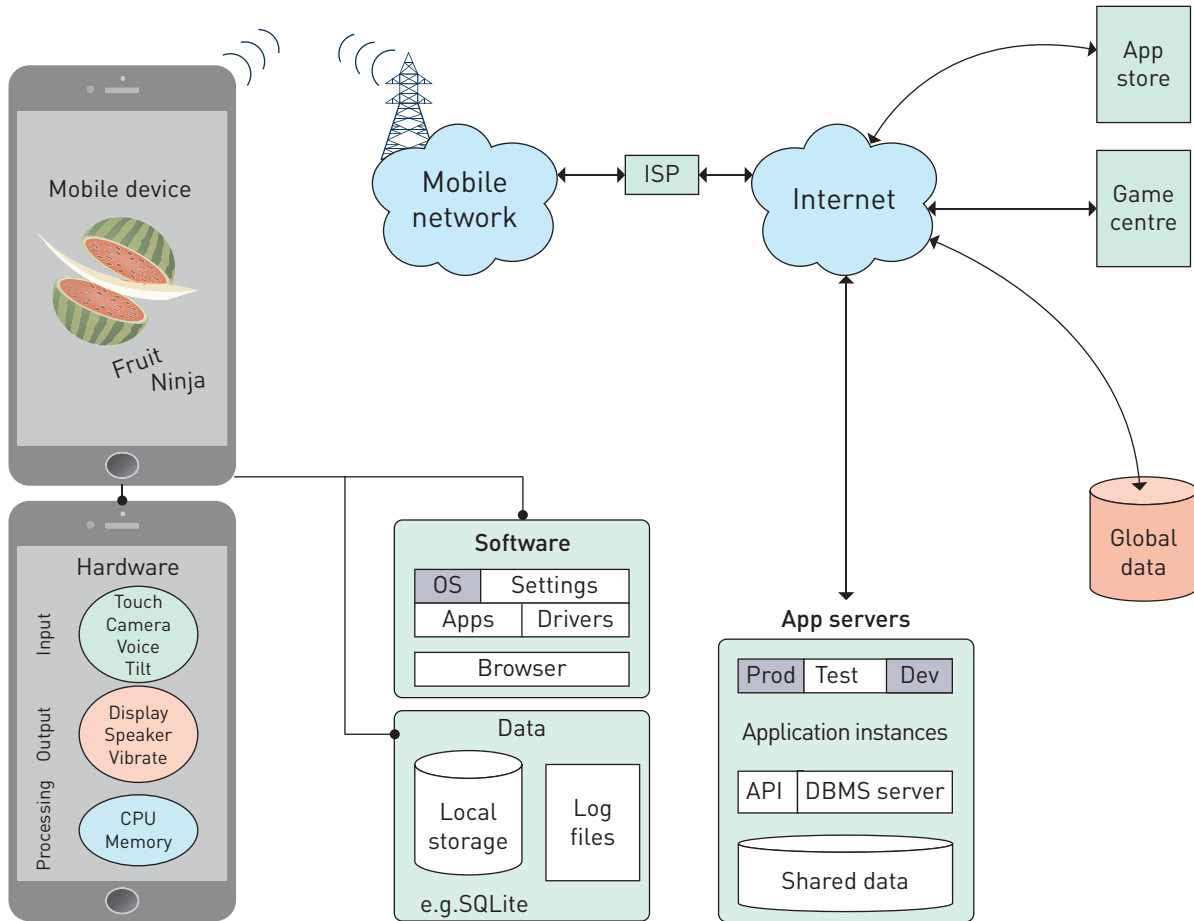


Figure 3.1.12 Elements of the Fruit Ninja mobile game

Many mobile applications and most prototypes are stand-alone. Once downloaded from an app store or development platform, they need no further interaction with the Internet or shared data. Other mobile applications share data with other users and need a constant network connection to work. Fruit Ninja began as a stand-alone game using local storage, but later versions require interaction with game centre data, shared leaderboards and in-app purchases.

When developing mobile applications or games, it is important to understand that not all mobile inputs and outputs can be easily simulated with a desktop prototype, and that the processing power and local storage on a mobile device is much more limited than on a desktop computer or laptop. While some testing can be simulated with bots or virtual devices, testing needs to occur on a mobile device before the application is published on the app store. Some mobile phone settings may need to be adjusted to enable features of the mobile phone that are usually disabled to conserve power.

The dimensions of mobile displays differ widely depending on the brand and model. Rather than creating a different version of the mobile application for every possible screen dimension, a library of media assets in



various dimensions are created for each mobile phone operating system. In this way, the appropriate icon or graphic can be selected for the particular device and used when the game is installed on that device. In web applications, a viewport setting can be used to adjust the scale of an entire web page with one line of code, but in mobile applications each version of the app must be provided with multiple sizes of every image used. For example, iOS apps must be published to the App Store with three different sizes of images, Android apps need six different sizes of image. There are a number of tools that can be used to automatically generate image collections in the required sizes, but some images do not downsample well. The designs of an iPad icon and an iPhone 5 icon, for example, may require some manual tweaking.

Programming for the user interactions with specific input and output components of a mobile phone relies on operating system-specific application programming interfaces (APIs). Interactions with local or external databases also require APIs. As discussed in previous units, an API provides programming language-specific functions that can be used to interact with hardware, software or data created by someone else.

User interactions

When developing for mobile phones, Halfbrick developers realised that they needed to build applications that would work on devices with limited memory and small screens, and that favoured touch input over keyboard input. At the time that Fruit Ninja was developed, most mobile phones focused on touch and tilt rather than swipe interactions. Today's mobile applications use a wide range of gesture controls, including tapping and pinching with two, three or four fingers, scrolling and rotating.

The prototype user interface for Fruit Ninja was developed in Flash on a PC with a mouse interface to simulate the swipe feature on a mobile phone. This was a good way to develop the concepts without the need for expert artists and programmers. Although the storyboard artwork did not initially appeal, the Flash prototype was much more popular. This shows that, when pitching an idea, it is important to use a tool that accurately conveys the mechanics of the system. Interactive prototypes are much more effective than images

Some mobile phone developers are already preparing for facial recognition controls in the future. They predict that future mobile games may be controlled with facial gestures such as a smile or a blink.



on a storyboard. There were some decision-makers who did not fully understand the market potential of the game until they played the game on a mobile device. Halfbrick also uses paper prototyping to test some game concepts, especially if the concept is too difficult to permit development of a digital prototype in a short time period.

The design decisions in Fruit Ninja focused on improving the user experience. The decision to use a plain black wooden board as the background in the initial release was based on the design principle of contrast. The contrast between the dark, plain board and bright, colourful fruit made the fruit more appealing and the fruit explosions more spectacular. The original peaceful sound background also contrasted against the sharp sound effects of slicing the fruit. The lead designer on the project asked his team to use design principles that emphasised the wetness and colour of the fruit, so that users felt as though they could smell the fruit.

Before release, the Fruit Ninja developers took their game to a bus stop and asked people waiting for the bus to test their game. They were deliberately seeking out people from their target audience. To ensure the game appealed to the broadest target audience possible, developers sought out people who had never played a game on a mobile touchscreen before. The test process was to simulate what would happen when users downloaded the game. They gave the downloaded game to the user without any instructions or prompts, and asked them to play it. During this first test with between 20 and 30 users, every single person handed the game back and said it didn't work. The concept of swiping the buttons to start the gameplay was so foreign that they couldn't figure it out. Something that was designed as a way to introduce the concept of the game became a stumbling block to actually playing the game. The feedback from this early user testing was used to refine the initial game screen, building in a graphical tutorial to detect if a user tapped the screen and show a swipe animation as a hint.

Later testing with expert users led to numerous suggestions to add complexity and interest to the game. Although it was tempting to use some of these suggestions, reflecting on the initial goal – to create a game that anyone could play – ensured that the simplicity of the game was preserved, and justified rejecting some

excellent suggestions for additional features. Ongoing user feedback for later revisions continues to be evaluated by the developers to identify potential new features to implement while still retaining the aspects of the game that make it popular.

Data processes, system interactions and control mechanisms

Halfbrick Studios uses a common game engine for all its games. The game engine is built almost exclusively in C++ and hosted on the Amazon Elastic Cloud (Amazon EC2). The Amazon cloud service provides the ability to create different test and production instances and migrate prototype software between different virtual machines. It includes a database server that contains all game assets, which can be shared across different platforms.

Although many game development companies use Objective C, Halfbrick chose C++ as its main programming language because it is compatible with a greater range of mobile devices and because there are many third party game-specific libraries that are suited to game development. Many game animations involve complex mathematics, so it is more efficient to use code libraries that have been developed and tested by experts than to calculate and implement complex mathematical algorithms from scratch. Code libraries contain reusable functions that make it much easier to implement game mechanics such as collisions, gravity and 3D shading. The integrated development environment (IDE) that Halfbrick developers use is predominantly Visual Studio, but Apple's Xcode was required for some iPhone and iPad applications. The Visual Studio IDE was selected as an organisational standard because it supports C++, HTML5 and many other programming languages.

Before Halfbrick Studios could develop mobile games, it needed to update its game engine to support mobile phone platforms and add event controls common to mobile phones. The first mobile platform to which it ported the game engine was iPhone. Each new type of phone needed a specialist build of the game engine, different sized application icons, backgrounds and driver interfaces. It was important to use a modular programming approach to make it easier to adjust the game engine for different platforms.

The fruit splats on the Fruit Ninja background image were coded as animations that transformed bullet-shaped objects into an explosion effect on the background to give the effect of fruit spray inspired by water droplets. Because this mechanic was coded with a modular approach, the programmer could adjust the code for each type of fruit using different parameters. Experimenting with different speeds for the trajectories allowed the best animation effects to be discovered.

As the game became more complex, automated testing bots were coded to collect game statistics on different variations of coded features. These bots have now evolved into artificially intelligent characters that gamers can compete with. Although the initial version of the game was simple, a database is required to store data for more complex game features. Mobile phone applications often use SQLite for local database storage, but Halfbrick uses a cloud-based database service.

Copyright considerations for interactive media

For the purposes of copyright, an interactive media app such as Fruit Ninja comprises many individual components, including software, graphics, scripts, animations, sound effects and background music. It is each individual component, rather than the entire interactive media solution, that is protected by the Australian *Copyright Act 1968*.

The fruit graphics in Fruit Ninja are protected as artistic works. The animations of the fruit splatter and the animated tutorial of the swipe action are protected as cinematograph films. The zen music in the background is protected as a musical work and the splat sounds of the fruit and explosion sounds of the bombs exploding are protected as sound recordings. The programming source code for the game mechanics is protected as a literary work.

In the Fruit Ninja example, all components of the app were created by employees of Halfbrick Studios. If a game incorporates work from other contributors, the studio needs to ask permission from each contributor and have written contracts in place, licensing the studio to use the content, before the interactive media app can be published and marketed. For this reason, most studios develop their own multimedia assets even if there is something similar available.



Some software libraries are published with a special type of license called an open source license. This means that the developer has given up their rights to the software and anyone is free to use it without negotiating with the developer for licensing rights. Multimedia assets are sometimes published with a Creative Commons licence and, provided the conditions of the license are met, no special licensing requirements are needed to use the assets.

CASE STUDY



Explore



3.1.6
Syropods:
CSIRO's
hexapod
robots

INTELLIGENT SYSTEMS

To understand innovative solutions within the intelligent systems technology context, the following pages explore a case study of an existing intelligent system to recognise the elements of the system and explain the interactions between all its components.

The intelligent systems case study is based on the CSIRO Syropod research project, which uses hexapod robots to navigate unmapped terrain.

SYROPODS

CSIRO uses its fleet of Syropods to traverse unstructured terrain. Small robots are used to explore confined spaces in the manufacturing industry and larger legged robots are used to collect data about rainforests. They are also helpful in exploring collapsed buildings or mines that are too dangerous for humans. Six-legged robots are inspired by nature and are modelled after insects. They can climb over obstacles without damaging crops or compacting the soil. Over a period of six years, CSIRO researchers developed efficient algorithms to select different walking patterns or gaits for different types of surfaces. Syropod robots autonomously detect when their gait is inefficient and switch to a more suitable walking movement. Although most robots use visual sensors to detect and interact with their surroundings, Syropod robots often need to operate efficiently in dark, foggy or smoky areas where visual sensors are not reliable. In addition to visual processing, Syropod robots use data from sound and position sensors to select the most appropriate movement algorithm for the type of surface they are walking on.

On concrete, it is faster and more efficient to move with three legs off the ground and three legs on the ground. On unsealed surfaces where legs can slip on loose dirt, leaves or sand, it is slower but better for stability to move in a wave gait with five legs on the ground and only one leg moving at a time. CSIRO found that it was impossible to select a single walking movement suitable for all surfaces, and recognised that it was necessary to train the Syropod robot to autonomously detect the terrain and adjust its walking style in the same way humans do. Legged robots can sense a change in terrain using a combination of strategies. They can compare the actual position of a leg placement with an expected, calculated position to measure the efficiency of a gait style. They can compare visual images of the terrain to visual maps of different types of terrain, or they can compare the audio features of the leg making contact with the terrain with stored features to identify when the robot transitions to a particular surface. CSIRO uses machine-learning strategies, such as a support vector machine, to train robots to recognise different types of terrain.



Figure 3.1.13 Legged robots can climb over obstacles more easily than wheeled robots.

© Copyright Navinda Kottege, CSIRO Australia. (<https://research.csiro.au/robotics/zee/>)

Elements of an intelligent system

An intelligent system has the following main components: actuators, sensors, controller, data storage and user interface. Actuators are the electronic components that take action, such as motors, lights and speakers. Sensors are the electronic devices that collect data, such as position sensors, light sensors and microphones. The elements of a hexapod intelligent system are shown in figure 3.1.14.

The hexapod robot uses three servomotors on each leg to control the coxa, femur and tibia leg segments. Each servomotor contains a position sensor. Early prototypes of the Syropod used an Arduino microprocessor to control the leg movements and sent data back to a laptop via a serial connection, to be logged in data files using a rosbag format. Rosbag log files can be processed in real time by Robot Operating System (ROS) software, or analysed and graphed offline using MATLAB programming language. Some Syropod prototypes use more powerful processing boards on board the hexapod to reduce the need for the laptop connection, but the laptop connection is still used for experimentation and testing of new algorithms in these advanced prototypes.

A microphone is mounted on the underbelly of the robot to detect the sound of the legs making contact with the terrain. A camera with a fish-eye lens was mounted to the top of the robot during testing to allow researchers to verify the terrain predictions made from the audio data.

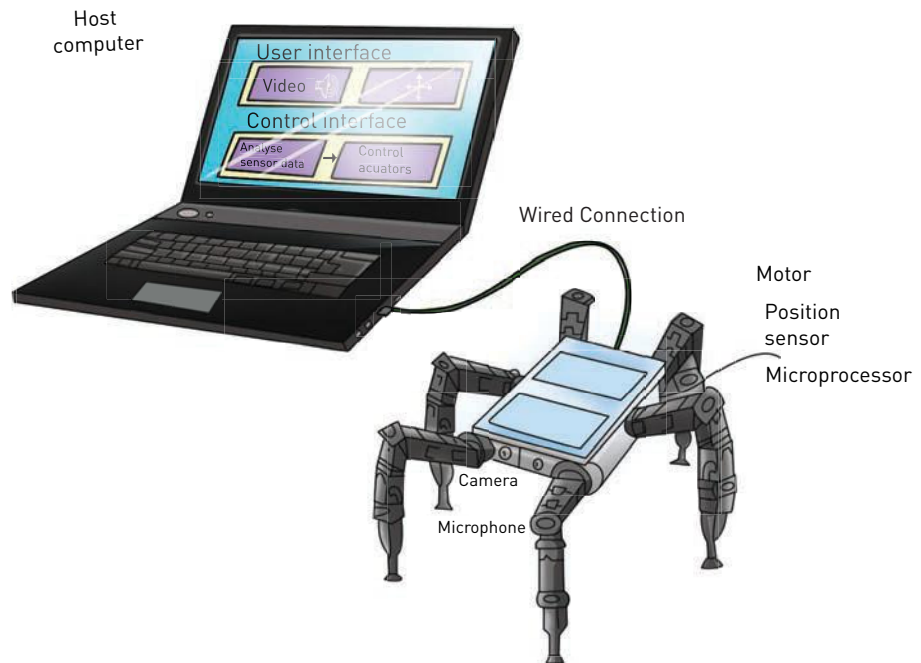


Figure 3.1.14 Elements of the hexapod intelligent system



3.1.7 ROS

Robot interactions

Robot interactions with different types of terrains were measured to collect data that could be used as a basis of detecting changes in terrain types. The terrains in figure 3.1.15 were used for testing to collect data about the way the robot legs responded in different conditions.

Robots were also held in the air so that a fifth category of data could be collected about leg movements where there was no friction. The test results showed that robots could not always accurately distinguish between rocks and mulch on leg position variance, so these two terrain types were sometimes combined into the same class.

The Robot Operating System (ROS) framework is a collection of libraries, tools and conventions that make it easier to create complex robotic software. ROS supports programming in C++, Python and Java, and there are many online resources and tutorials available. It also works with the Gazebo robot simulator, which is an open source tool used to test robot designs, algorithms and autonomous behaviour in realistic simulations of indoor and outdoor environments. ROS is used by almost all robotics researchers and is often listed as a pre-requisite skill for robotics-related jobs.



3.1.8 Gazebo
robot simulator
website.

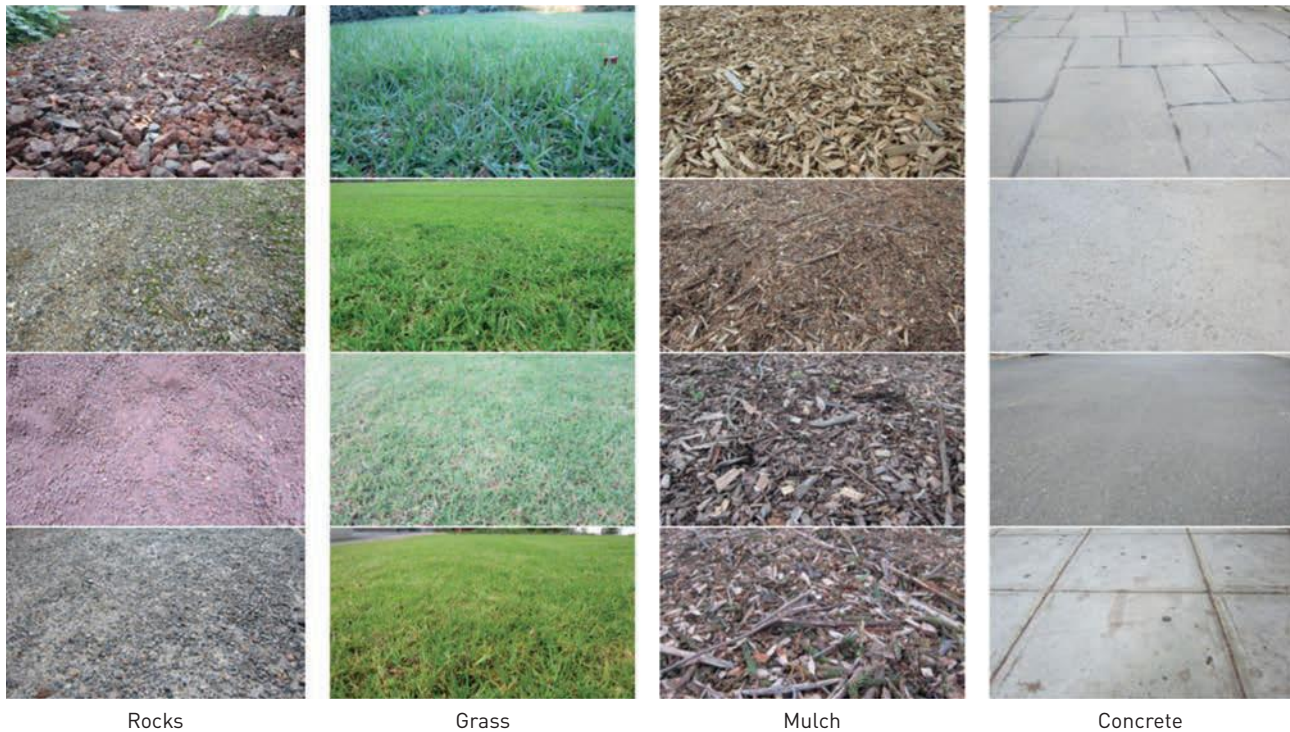


Figure 3.1.15 Four types of terrain were used for testing the terrain classification algorithm.

A separate research team used audio sensors to detect the terrain, and they included asphalt, ceramic tiles and carpet as additional test surfaces. In the audio trials, asphalt, concrete and tiles generated similar audio characteristics, as did grass, mulch and gravel. Extensive manual processing of audio data was required to eliminate random background noises from audio samples, such as cars and birds. Constant noise such as that of the servomotors was eliminated with spectral subtraction. These adjustments to the audio samples improved the performance of the audio data algorithm in determining changes in terrain.

Data processes and system interactions

The walking gait algorithm used to detect a terrain change is:

Turn on data sensing and logging

REPEAT

Take 3 steps

Synchronise data using known timestamps and interpolation to predict missing data

Subtract goal position from actual position for each servo

Determine terrain class using stored data

Select appropriate gait algorithm

UNTIL stop signal received

The data collected during testing was referenced by this algorithm to determine the appropriate terrain class, as shown in figure 3.1.16.

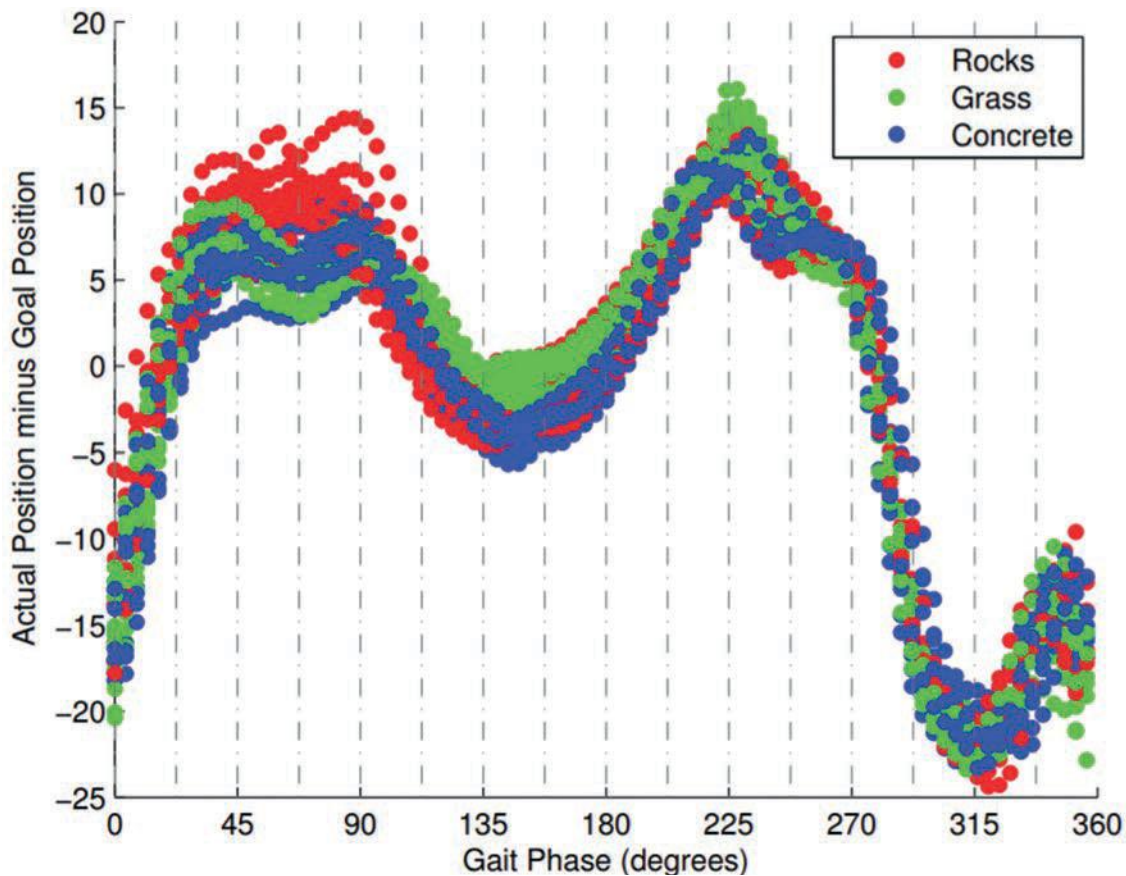


Figure 3.1.16 The difference between actual position and expected position of the robot legs can be used to determine the terrain class.

Control mechanisms

The control system CSIRO uses to program all its robotics equipment is the Robot Operating System (ROS), which supports a range of programming languages and database interfaces. Using the gait selection algorithm, the robot was able to autonomously select the most appropriate gait style 95 per cent of the time. The results of one test are shown in figure 3.1.17.

The ROS software also includes an open-source plug-in called HARK-ROS, which was used for collection and processing of audio data from the microphone sensor located underneath the robot. Fly Capture open source software was used to process images.

This research project was conducted by professional scientists and PhD researchers, so the mathematical and scientific concepts used are very complex. The aim of CSIRO research projects such as these is to develop algorithms and software libraries that allow programmers without specialist knowledge to use these approaches. Similar software libraries are available for use by students for other types of intelligent systems.



Figure 3.1.17 When the terrain detection algorithm was implemented in ROS software, this terrain was classified as a combination of grass (green), rocks (blue) and mulch (red).

Applying systems thinking to explore digital solutions

Innovative solutions involving multiple components can sometimes have unintended consequences. Systems thinking helps us to understand the potential impacts of an innovative solution. Systems thinking is a process for understanding the components of complex systems, how the various parts of the system interact with other parts and the impacts of the system on external entities and other systems. It is used in a range of disciplines, including business, education, sociology, science, design, technology and engineering.

When applied to the technology context of a digital solution, systems thinking can help to:

- understand the interactions between components of a digital solution
- understand the cause/effect of changing a component in a digital solution
- understand the scope of a problem
- identify critical and optional features of a digital solution
- analyse inputs and outputs to identify errors and inefficiencies in a digital solution
- identify or predict possible social and ethical impacts on external entities and systems.

There are many different ways to apply systems thinking to a digital system context. The main idea is to focus on the interconnections between different parts of the system and understand that a change to one component can affect other, related components. Systems thinking requires circular, rather than linear, thinking as the interconnections between components of the system are usually messy and ill-structured. Complex systems cannot be represented as a linear sequence or hierarchy.

Causal loop diagrams, behaviour-over-time graphs and connected circles are all examples of systems thinking tools used to explore established complex systems. Some of these diagrams can be helpful in problem-solving technical problems, but not all are suitable for developing innovative solutions. One popular graphical tool that software developers use to represent a system is the concept map or concept diagram. Components and interactions of a complex digital solution can be easily and quickly represented as concepts and connections in a concept diagram. This allows the developer to quickly and easily document early requirements of a digital solution from the user and developer perspectives, and online tools make it easy to adapt the diagram to accommodate regular changes as the requirements and designs evolve throughout a project. Using systems thinking to analyse a digital solution represented as a concept diagram can also be helpful when solving problems or understanding the cause of unexpected behaviours between the user interface, data elements and programmed components.

Creating a concept map of a digital solution

The first step of the systems thinking process is to identify the possible parts of the system. The quickest way to do this is to speak to someone who understands the system (or a similar system if it is still in development) and ask them to tell you the story behind it. As each part of the system is mentioned, a concept bubble is drawn to represent it. Interactions between parts of the system are shown with lines or arrows connecting the concept bubbles. Another way to identify parts of a system is to observe a system in action and draw a concept map of what you observe. This will not give you the complete picture of the internal components, but it will help you understand the external interactions between the observable parts of the system and their influence on other systems.

Systems can be closed or open. Closed systems are self-contained, whereas open systems interact with other systems or entities. Very few digital systems are closed systems since most digital equipment requires user interaction. In a digital solution, any external entities (such as the user) that are outside the control of the developer should be considered to be outside the system boundary. Figure 3.1.18 shows a simple diagram of an interactive media system. Users interact with, but are outside, the boundary of the system. The gaming interface is a separate system and is also outside the control of the developer.

Concept maps are not a traditional systems thinking tool, but they can be used effectively to map the components of a system as a starting point for systems thinking. Concept maps have the capacity to communicate a complex story in an easy-to-understand diagram. Although some developers choose to use

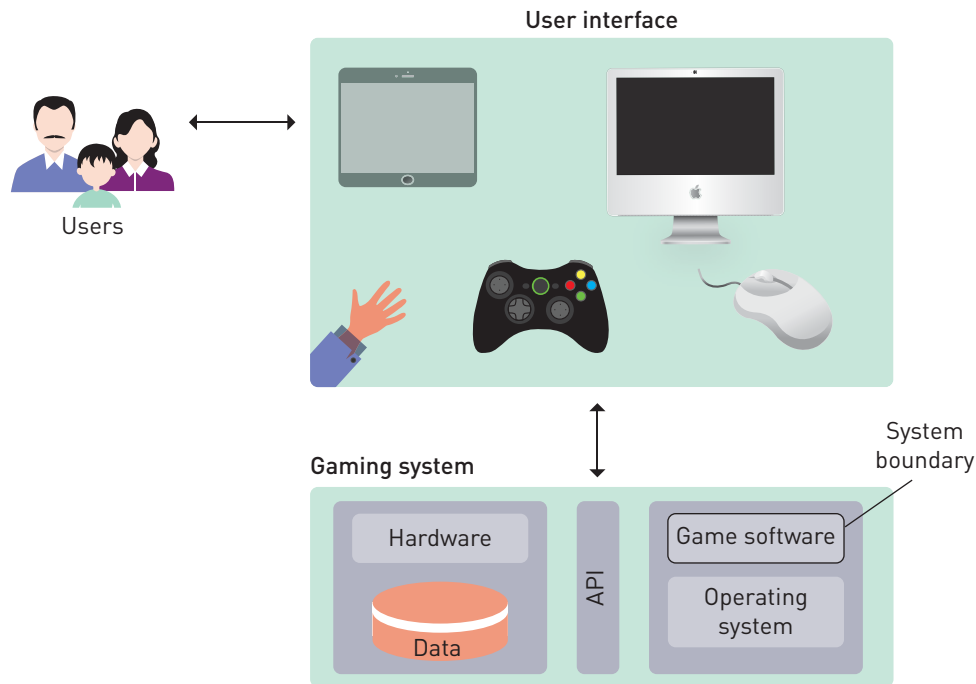


Figure 3.1.18 The system boundary should be drawn around components that are within the control of the developer.

Unified Modelling Language (UML) or other flowchart symbols in digital concept maps, it is not necessary to use special symbols or notations. A simple bubble shape can be used to represent each concept (or component) and an arrow can be used to show interactions.

Concepts are the data, users or programmed components of the system and can be represented as a simple bubble shape. Connections show the inputs and outputs of the system and are represented as an arrow. A concept map is built by using connections to show interactions between components of the system.

Closed connections have a concept at each end of the arrow. Open connections have a concept at one end, but the other end of the arrow is open. The boundary of the system includes all the concepts and closed connections. Open connections show the external inputs and outputs outside the boundary of the system. If there are no open connections in a system, it is a closed system. Figure 3.1.19 shows how concepts, connections and system boundaries can be used to represent open and closed systems.

When constructing a concept map of a digital solution, some components may initially be missing or incorrect. It is important to revisit the systems thinking process regularly to add additional components as they are discovered, and gradually build an accurate map of the entire system. Using a digital or online tool to construct the concept map makes it easier to adapt to changes or corrections without having to redraw the entire map.

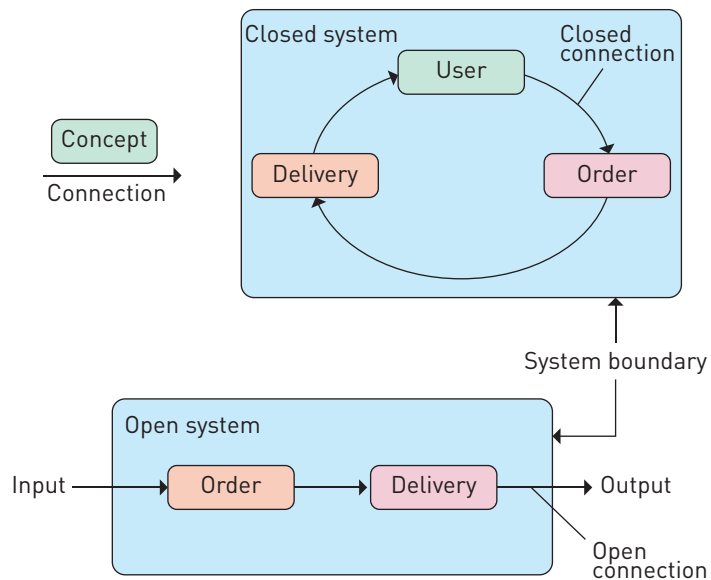


Figure 3.1.19 Concept maps show the connections between components of a system.



Tools suitable for creating a concept map include:

- Microsoft Visio
- SmartDraw
- Bubbl.us
- Lucidchart
- Creatly.com
- Coggle.it
- Loopy
- Clickcharts.

When all the concepts and connections are mapped, annotations that describe interactions in more detail can be added. ‘What if’ questions help to predict how a user might interact with a digital solution and how expected and unexpected input will be managed. ‘Why’ questions help the developer understand the reasons for specific concepts and connections.



ACTIVITY 3.1.3

Choose a digital or online concept mapping tool and create a concept map that shows all the different ways students travel to school.

- 1 Around the outside of the concept map, include an image shape or text box to represent each mode of travel, for example bus, car, bike, walking.
- 2 Towards the centre, create a concept bubble for each different entry point at the school, such as car park, bike rack, main entry.
- 3 Draw connection points to show the typical connections. For example, a student who rides a bike might enter the school near the bike racks. This can be represented on the concept map by drawing an arrow from the bicycle to the bike rack concept. Consider that some students may combine different modes of travel to school.
- 4 Test and refine your concept map by checking it has connections for the following examples:
 - A student arrives by car with their parent, who is a teacher at the school.
 - A parent drops a student off at the train station. The student travels on the train to the train station closest to the school and then walks 800 metres to enter the school at the main gate.
 - A student rides a bicycle to a friend’s house, leaves the bike at the house and walks 250 metres to enter the school at the main gate.
 - A student rides a bicycle to school, chains it to the bike rack and then walks the rest of the way, entering the school through the side gate.
- 5 Look for patterns in the concept map and ask questions to understand the patterns. For example:
 - Which school entry point has the most connections?
 - Why would a student lock their bike in the bike rack and then enter through the main gate?
- 6 Consider how a concept map like this could be used to map user pathways through an application. Which symbols would represent the web pages? Which symbols would represent types of users?
- 7 Resize the concept bubbles in the school travel concept map based on the number of incoming connections, and the number of students each connection represents, to show the busiest entry points as large bubbles and the less busy entry points as smaller bubbles.
- 8 Consider how systems thinking could be used to find and eliminate bottlenecks in a digital solution or identify the most important part of a solution to build first.

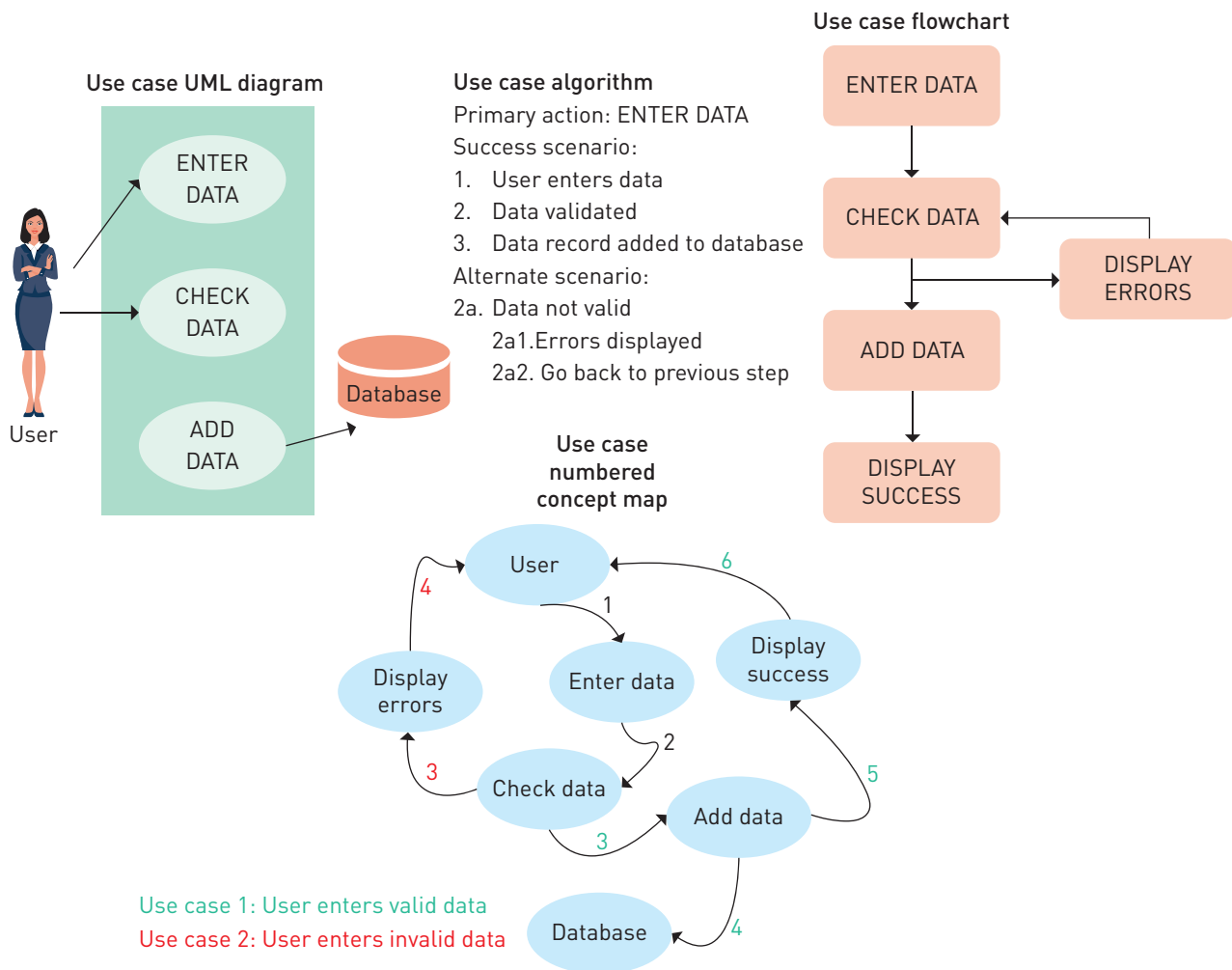


Figure 3.1.20 Use cases can be represented as UML diagrams, algorithms, flowcharts or numbered concept maps.

Understanding user expectations with use cases

A use case is a step-by-step description of a scenario whereby a user interacts with a digital solution for a specific purpose; for example, the steps carried out by the system to enrol a student in an online course, or a description of what happens when someone updates their social media profile. A use case is usually represented as a short paragraph describing the process from the user perspective, but it could also be represented as a flowchart, a formal UML use case diagram, an algorithm, or by numbered annotations on a concept map (see figure 3.1.20). Formal use cases are not a syllabus requirement for Digital Solutions, but a numbered concept map is a useful systems thinking tool that can help to explain, symbolise and analyse user interactions with a system.

Systems thinking can be used to analyse typical use cases for a proposed system to determine the desired behaviour or requirements for a solution and the impacts of system failures. Basic use cases are often referred to as ‘sunny day’ use cases that describe what happens if everything works correctly. Considering ‘rainy day’ or alternate use cases can help to build resilience into a system – to plan how to manage a scenario where parts of the system fail.

When changes are recommended for a system, systems thinking can be used to analyse the impact of introducing the changes. Identifying the system components affected by the change helps in understanding the scope of the change. Planning the change across the entire system, instead of in one part of the system, reduces the likelihood of introducing errors in other parts of the system.



GROUP ACTIVITY 3.1.1

Consider the user experience requirements for a digital solution that allows online purchasing of movie tickets.

When you purchase a movie ticket online with a movie gift card, you expect money to be debited from your gift card balance and that there will be a ticket available for you to collect from the cinema. You may also expect that your rewards card balance will be checked first before debiting funds from your movie gift card because you are entitled to receive your 10th movie ticket free under the conditions of the reward club membership. You expect the system to provide accurate information about the movies showing at the cinema on the date and time you want to purchase the ticket, and you don't want to purchase a ticket if the session is sold out. You expect the system to let you know if your purchase was successful or unsuccessful and send you details of how to collect your ticket, such as an email containing a printable ticket, registration number or barcode. To do all this, the system may need to be regularly updated with movie session data, gift card balance data and rewards card data by administration staff or online systems outside the boundary of the online movie ticket system.

- 1 Form groups of two or three students and use poster paper or an online concept mapping tool to create a concept map that shows the information about the digital solution described above. It will probably look something like the diagram in figure 3.1.21.
- 2 Refine and annotate the concept map to represent how the following use cases could be handled by the system:
 - A friend wants to see the movie with you and purchases a ticket at the cinema.
 - The proof of purchase is emailed to the wrong account.
 - Two people try to reserve the same seat at the same time.
- 3 The manager of the cinema wants to introduce a new reward system for local cinema members that allows them to watch free additional screenings of any movie they purchase a ticket for. As part of this promotion, all rewards card members will receive a new membership card with a special membership ID.
 - a Update the concept map to show how this feature could be incorporated into the current system.
 - b Write use cases for a customer who is a member and a customer who is not a member.
 - c Test the use cases and identify the parts of the system that will be affected by the change.
- 4 Reflect on the systems thinking process you used for this activity. Was the systems thinking process helpful in understanding the components of the system and the impact of changes to the system?

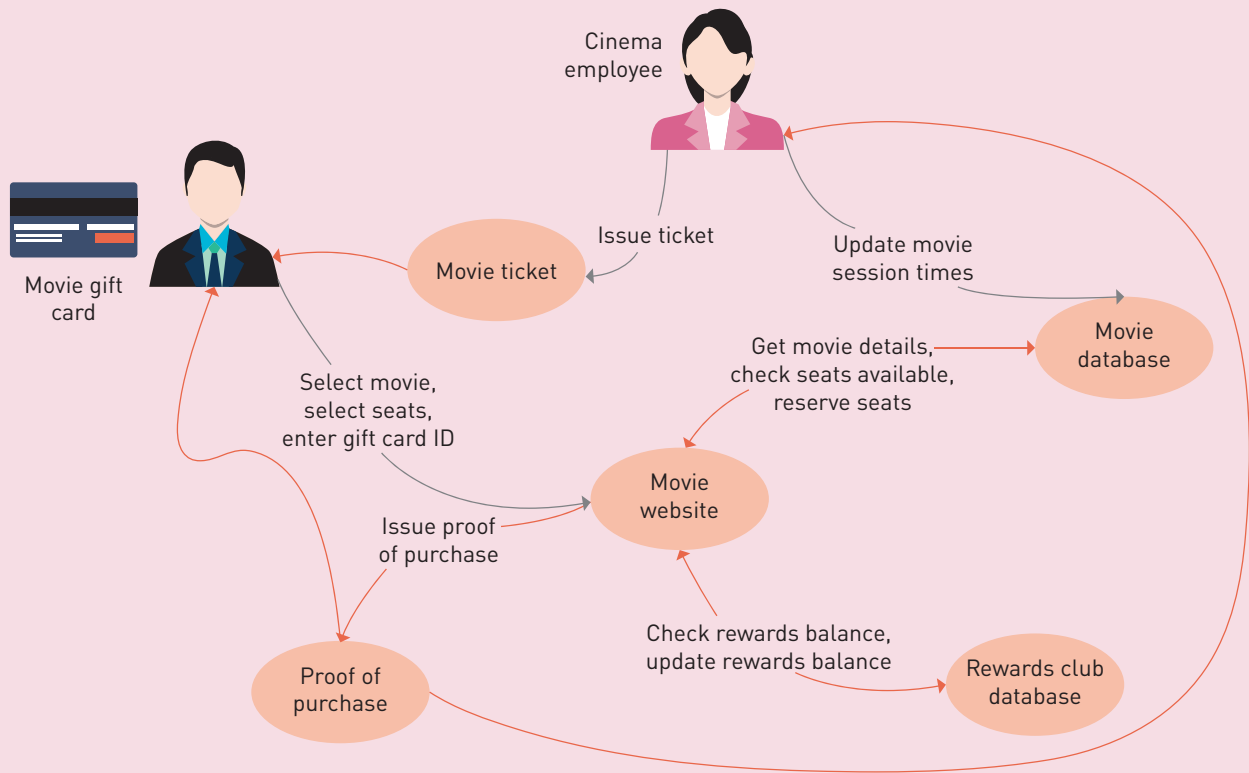


Figure 3.1.21 An example concept map for the box office rewards problem

Using systems thinking to resolve problems

Most problem-solving processes look at the problem effect and look for one root cause. Systems thinking recognises that one problem can have multiple causes. To solve a problem in a complex system, it is important to consider all the inputs and outputs related to the problem and consider all the possible causes. If the problem is intermittent, a behaviour-over-time graph may help to identify unexpected events that are contributing to the problem. Asking what-if and why questions will help to focus the investigation in a particular area of the system. Collecting input and output data and looking for problematic patterns can help to eliminate or verify contributing factors.

- 1 Create a map of the problem system.
- 2 Collect input and output data and look for patterns.
- 3 Develop focus questions that identify possible problem components and areas for further investigation.
- 4 List possible causes and consequences.
- 5 Look deeper into the problem to eliminate or verify contributing factors.
- 6 Make small changes to attempt to resolve the problem.
- 7 Continue to monitor the system and make additional changes if required.

ACTIVITY 3.1.4

Describe how systems thinking could help you to find the cause of *one* of the following problems:

- An application always crashes at a particular time of the day.
- A robot vacuum cleaner always misses the same corner of the room.
- Profile data for the same user is different on two different devices.



TEST YOUR KNOWLEDGE

Exploring a digital solution

- 1 List the essential components of a web application.
- 2 Describe four different user interfaces.
- 3 Describe the role of an actuator in an intelligent system.
- 4 What type of diagram is helpful in symbolising the processes and data flows in a digital solution?
- 5 Explain what is meant by the term 'user experience'.
- 6 What is a PHP server?
- 7 What is the role of CSS in a web application?
- 8 In the context of programmed components of a digital solution, what is meant by the term 'library'?
- 9 Describe two activities that benefit from systems thinking.
- 10 List two types of user input or output available on a mobile phone that are not available on a standard laptop interface.
- 11 How can the systems thinking process be used to desk check a solution?
- 12 How does the lack of standardisation of mobile phone screen sizes impact the development of mobile phone applications?
- 13 Using one of the case studies explored in this section, explain how prescribed and self-determined criteria can be used to measure the success of a digital solution.
- 14 Explain why interactive media companies take the time to develop their own multimedia assets.
- 15 Explain the two main differences between prescribed and self-determined criteria.
- 16 Being the first indie company to develop a game with an innovative feature can be an ingredient for success, but it can also be a huge risk that jeopardises the future of the company. Use systems thinking to analyse the risks of using emerging technologies such as facial recognition as a required component for an interactive media application.
- 17 Summarise what you learned from one of the case studies in this section.
- 18 Write a sunny day use case for the Koala Tracker web application.
- 19 Write a rainy day use case for the Fruit Ninja mobile game.



APPLY YOUR KNOWLEDGE

- 1 Follow the systems thinking process to create a concept map for the automatic watering system described below:
A data sensor collects moisture data from the soil at various times throughout the day and transmits it wirelessly to a web server, which then records the data with a time stamp. At 6 p.m. each night, the server is scheduled to run a program that analyses the moisture data logs and emails a report showing which plants need to be watered. In the morning the lab assistant checks the email and waters the plants that need to be watered.
- 2 Analyse the interactions in the concept map to determine the expected inputs and outputs for the following use case:
There are five plants with new data sensors installed that are all operating correctly, the wireless network is operating efficiently and the report is emailed to the correct person.

- 3 At the end of the month the scientist checks the plants. One of the plants is overflowing with water and is scheduled to be watered again that day. Another plant is completely dry and has not been watered all week. All the other plants are flourishing and have the correct moisture levels.

Analyse the interactions in the diagram to determine the expected inputs and outputs for the alternate use cases below:

- a One of the sensors has fallen out of the plant and is not operating correctly.
- b The wi-fi capability on one of the sensors has been disabled.

To what extent do the alternate use cases explain the overwatering and underwatering issues?

- 4 Propose changes to the system that could address the issues identified.

3.2

DEVELOPING AN INNOVATIVE SOLUTION

In this section, you will use design thinking to explore innovative ideas for problems that are relevant to you and analyse problems to determine the user, programming and data requirements of possible solutions. You will design the individual parts of possible solutions and use diagrams to show how the parts will work together to solve the problem.



Analysing innovation

Innovation is the art of finding a better solution to a new or existing problem. Digital innovation is a specific type of innovation that involves digital technology. Some successful digital solutions become so widespread they are referred to as **digital disruption** because they have such a significant impact. As shown in figure 3.2.1, digital innovation has the power to improve the lives of people, communities, businesses and even entire countries.

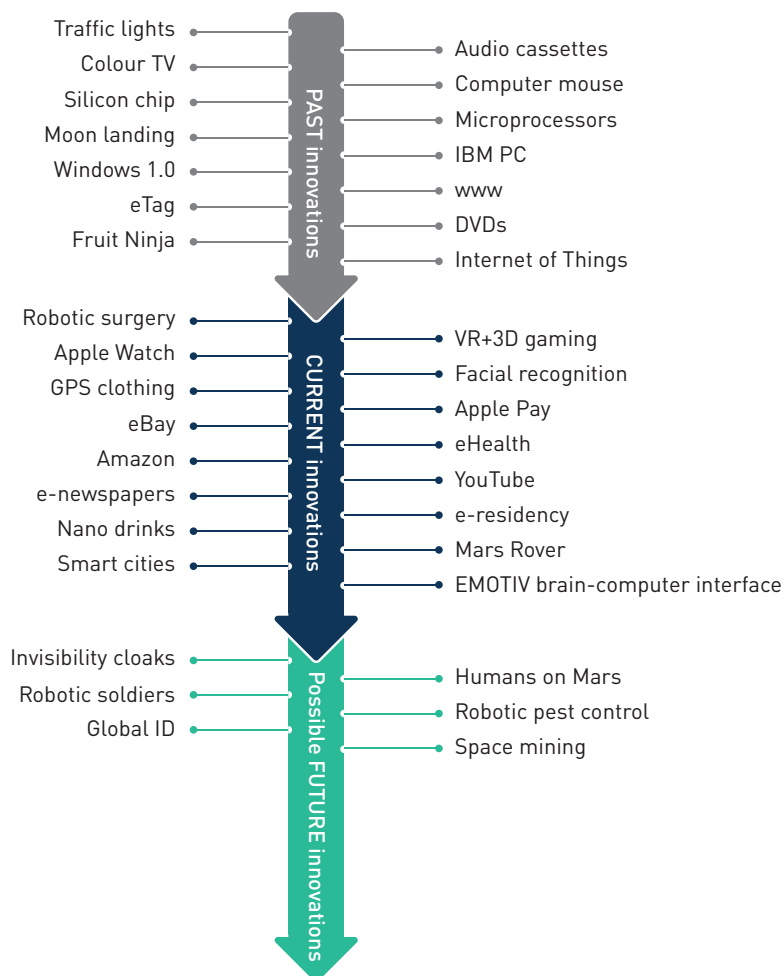


Figure 3.2.1 Digital innovation improves the lives of people, communities and businesses.

Using design thinking in a digital technologies context

Design thinking is used to brainstorm innovative approaches. It is successfully used in user experience design to transform the way people interact with technology because the target user is the central focus of the design. Design thinking encourages creative brainstorming that focuses on the interests and needs of people and generates lots of possible solution ideas. It then applies critical thinking about environmental, technical, social and economic constraints to focus on the most important parts of the problem. Design prototypes developed using a design thinking process solve relevant problems, sometimes in unexpected ways.

‘Design thinking is a human-centred approach to innovation that draws from the designer’s toolkit to integrate the needs of people, the possibilities of technology, and the requirements for business success.’

Tim Brown, President and CEO of IDEO

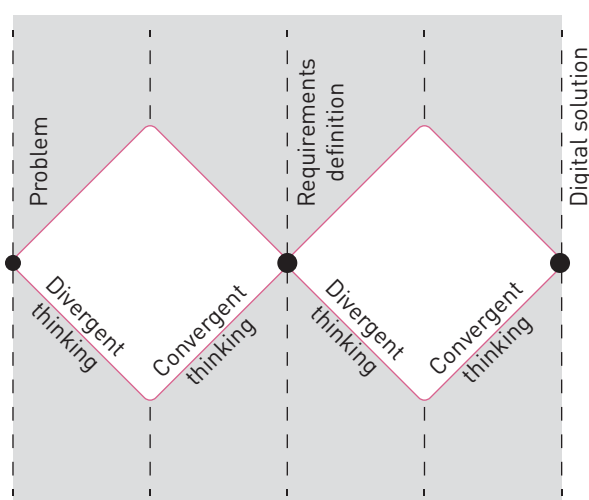


Figure 3.2.2 Design thinking is an iterative process of divergent and convergent thinking.

Design thinking is an iterative process of convergence and divergence and is commonly represented as a double diamond, as shown in figure 3.2.2.

The process begins with a problem and the people who are affected by the problem.

A team of design thinkers brainstorm the problem, considering as many different angles as they can. This is referred to as **divergent thinking**. During divergent brainstorming it is important to add as much information as possible in as short a time as possible, regardless of how improbable or irrelevant the information may seem.

After a set time, the team moves to **convergent thinking**. This involves organising and refining the information to identify common themes, features or issues and to define a critical aspect of the problem to focus on.

The team then switches back to divergent brainstorming about a range of possible solutions to the problem. It is important to add as many new ideas as possible in as short a time as possible regardless of how impossible or impractical the ideas may seem. During divergent brainstorming it is helpful to enforce a few simple rules to ensure the ideas flow freely to create as many options as possible.

DESIGN THINKING GUIDELINES

- 1 Defer judgement – there is no such thing as a bad idea.
- 2 Aim for quantity – write down every idea.
- 3 Go wild – encourage crazy ideas.
- 4 Build and combine – always ‘AND’, never ‘BUT’.

The next step is to use convergent thinking to evaluate each option and choose the best option. Convergent thinking takes more time than divergent thinking as it is important to carefully consider each idea (no matter how crazy it sounds at first) and judge it against specific criteria relevant to the problem you are trying to address. During this process, constraints, limitations and criteria are introduced to sort the possible from the impossible.

An innovative design thinker needs to have an open mind and to be willing to consider the ideas of others. Collaboration is a key ingredient to innovation, as one person's crazy idea may lead a design in an unexpected direction and trigger other, more realistic ideas.

'The only way to discover the limits of the possible is to go beyond them into the impossible.'

Arthur C Clarke

Design thinking is commonly applied at the start of a project, but is most successful when used iteratively in a repeating cycle of divergence and convergence. This means that the process is repeated many times. The failure of one idea leads to new insights that feed into a new, divergent brainstorming session about the problem that then converges into a new idea that can be prototyped and tested. Each iteration of the design process brings new insights into the problem and better understanding of the limitations of technology, which can lead to more effective designs and more efficient prototypes.

Design thinking can be used in development of innovative digital solutions when we:

- brainstorm ideas at the start of the project
- imagine how different people will interact with the solution to improve the useability
- solve a difficult problem or roadblock in a project
- work around constraints and limitations using limited equipment and skills
- imagine new uses for old technology
- consider future uses or impacts for current technology.

Group activities

These group activities are designed to develop design thinking skills for digital innovation.

GROUP ACTIVITY 3.2.1

Design thinking helps us unlock our creativity

In figure 3.2.3, an artist is looking for a needle in a haystack, one strand of hay at a time. Is this the best solution to the problem? Can we come up with a better solution using design thinking?

When we use design thinking, we need to remember the design thinking guidelines. For example, if one of the ideas about finding a needle in a haystack was to call 000, would you write down the idea? The answer is yes, because there is no such thing as a bad idea and you need to write down every idea. It is especially important to write it down because it is a crazy idea, and other people might build on this idea and think about someone else they could call.

Form a group of between four and six students. In your group, use design thinking to think about some creative and different ways to remove a needle from a haystack. Set a short time limit (no more than 5 minutes) and nominate a timekeeper. When the timekeeper tells you to start, brainstorm as many ways as you can to remove a needle from a haystack. After the time limit, share your craziest idea with the rest of the class.

How could this process be used to generate ideas for an innovative project idea?

Use design thinking to brainstorm solutions for the following problems:

- What is the best use for a paperclip?
- How can I remember to charge my mobile phone?
- What can I make with pipe cleaners and playdough?

Reflect on your strengths and weaknesses during this activity. What will you do differently next time?



Figure 3.2.3 Sven Sachs hunts for a needle in a haystack as part of a performance art exhibit in Paris.



Develop



Evaluate

REUTERS/Charles Platiau



Develop



Evaluate

GROUP ACTIVITY 3.2.2

Design thinking helps us unlock the power of empathy

The pictures in figure 3.2.4 represent different target audiences in the local community. Using one of these pictures for inspiration, you will create a user persona and then use design thinking to brainstorm a mobile app idea to help the fictional user created by one of the other groups.



Figure 3.2.4 Each of these four photos represents a different type of user.

Here is an example of a user persona.

Name: Billy

Billy's backstory: Billy (see figure 3.2.5) is the youngest of eight children. His family has a cat and a dog. There are two desktop computers, two laptops and three tablet devices in his house. When his brothers and sisters are at school, he is lonely and bored. His mum works part-time. There is a computer at the childcare centre. He has no computer skills. He can sit well but has trouble communicating with his parents and is not yet able to walk.

A persona is a representation of a user that helps designers identify with the target audience. Personas are created from:

- research and analysis using demographic data
- statistical information about groups of customers
- interviews with real customers.



Figure 3.2.5 Photographs help developers to empathise with the target audience.

Clockwise from top left: iStock/Getty Images Plus/
AntonioGuillem; Getty Images Plus/E+/sdmnick;
iStock/Getty Images Plus/Antonio_Diaz; Getty Images
Plus/E+/fotostorm

iStock/Getty Images Plus/goodmoments

Proto-personas are created using design thinking, including:

- educated guesses
- role-play
- mock interviews.

Practise creating personas.

- 1 Form a group of between four and eight students and select one of the sample pictures. In your group, spend 5 minutes brainstorming the backstory for the person or people in your photo. (Note: this activity will work best if each group chooses a different picture.)
- 2 Spend 5 minutes brainstorming five questions to ask another group that will help you discover the backstory behind their photo (your new client).
- 3 Nominate one group member to take on the role of the person in your photo and send them to another group as a client representative. In return, your group will host a client representative from one of the other groups.
- 4 Using the five questions brainstormed earlier, each group interviews the visiting client representative to discover the backstory for their client's persona. The client representative answers the questions using the backstory created during the first brainstorming activity. It may be necessary to use role-play and creativity to extend the backstory during the interview.
- 5 When all the visiting client representatives have finished answering the five questions, each representative should return to their original group.
- 6 The rest of the group should then brief the returning group member about the backstory for the visiting client representative. This is the client they need to consider during the next part of the group activity. Use the personas you have developed in the next part of the activity.
- 7 In your original group, spend 10 minutes using design thinking to brainstorm a new mobile phone app that solves a problem for the client your group interviewed. For example, a solution to Billy's problem about being lonely and bored might be an app that lets him touch a picture of a toy, which triggers a robot to deliver the toy to him.
- 8 Use poster paper or a digital collaborative environment to contribute ideas. Each student should use a different coloured pen or tool to contribute ideas. There should be an equal quantity of each colour on the paper or online representation of the idea to show that all group members have contributed.
- 9 When your 10 minutes are up, stop work. Pitch your idea to the class and listen to the pitches from other groups.

In a class discussion, compare this design process to other approaches you have used in the past. What are the benefits of using design thinking? What are the risks?

Reflect on the benefits and risks of using design thinking for a school project. How can you harness the benefits and mitigate the risks in your next project or activity?

GROUP ACTIVITY 3.2.3

Open data design challenge

Each year, universities and other organisations throughout Australia host GovHack, a 48-hour HackaThon that encourages innovative use of open datasets. Use the weblink below to watch an award-winning pitch about a prototype of an innovative chatbot for use by visitors to the Gold Coast during the 2018 Commonwealth Games.

What innovative web applications could you create using open data?

Open Data Institute Queensland has a collection of open datasets available for download by individuals and organisations. Their mission is to encourage innovation using data. ODI Queensland provides a list of open datasets available from Australian websites. Datasets can be downloaded in many different formats. For this exercise, you will use environmental data from Brisbane City Council in CSV format. A CSV file is a sequential file that contains data values separated by commas. You can open a CSV file using Microsoft Excel to see the data in columns and rows. During the next activity you will use divergent



Develop



Evaluate



3.2.1 GovHack
3.2.2 Open
source data



thinking in small groups to brainstorm innovative use of open datasets, and convergent thinking as a class group to select the most useful ideas.

Practise using design thinking in a group of three or four.

- 1 Access three Brisbane City Council open datasets in CSV format. These can be downloaded, previewed online or provided by your teacher. Using only the data available in the data samples, brainstorm ideas for a web application that could make use of this data to improve the lifestyle of Brisbane residents.
- 2 After 10 minutes, share your ideas with the rest of the class.
- 3 Use convergent thinking to evaluate each idea as it is shared. Give each idea a score between 0 and 9 using the following criteria:

CRITERIA	RATING	SCORE
To what extent will the idea improve lifestyles of Brisbane residents?	Helpful for most residents	3
	Helpful for some residents	2
	Helpful for a few residents	1
	Not at all helpful	0
How much of the data is available?	All data is available	3
	Some data is available	2
	A few data values are available	1
	Dataset is not used	0
Is this possible with current technology?	Yes	3
	Mostly	2
	Partly	1
	Not at all	0

- 4 Does the class agree on the rating for each idea? Discuss any differences of opinion to obtain agreement on the correct rating for each idea.
- 5 Using the agreed ratings, identify the most useful proposals.
- 6 Form new groups for each useful proposal and spend an additional 10 minutes using divergent and convergent thinking to refine the initial proposal.
- 7 As a group, pitch your idea to the rest of the class and listen to their suggestions. Provide suggestions to other groups about additional features they could include.

In a class discussion, review the design thinking double diamond model and map the steps in the group activity to the design thinking process.

Reflect on how the design thinking process influenced the design. How could this process be used effectively in a school project?



3.2.3 Brisbane City Council open datasets



Explore

ACTIVITY 3.2.1

- 1 Explore the term 'digital disruption' and describe an example of digital disruption that has impacted you or your community during the past decade.
- 2 Research new technology that is predicted to disrupt business in the near future and use design thinking to brainstorm ways in which this technology could be used to improve your life in some way.



3.2.4 A view into digital disruption
3.2.5 Digital disruption - opportunities for innovation and growth

Defining the problem

At the beginning of a new project, it is very tempting to jump straight in and start creating the different components of the solution. However, unless you have some idea of what the customer needs, you could spend a lot of time creating different parts that are never used in the final solution. You may even run out of time to complete the project.

As discussed in Units 1 and 2, determining the requirements for a digital solution involves identifying the **functional requirements** (what the solution needs to do), the **non-functional requirements** (how it needs to do it) and the **constraints** of the user and developer environments. Together, these requirements will become the prescribed criteria for your digital solution.

Functional requirements are a specific list of actions that will be controlled by code. For the user interface, this could be specific equipment to be used, a description of how the user will interact with the solution and specific text or data fields the user will see on a visual display. For the data interface, this will focus on specific datasets and their file formats and data structure. It may also include any specific requirements to clean the data (remove errors), restructure or filter existing data, and methods that will be used to collect new data required for the solution. For the coded components, this will probably be a list of specific code modules that need to be developed to process user input, calculate using data, and store or display data.

Non-functional requirements are specific features about how the coded components should work. They include legal, economic, environmental and aesthetic requirements that will influence the development of the solution. Legal considerations can include the Data Privacy Act, copyright, intellectual property and existing contractual arrangements. Economic considerations will be the cost of creating and maintaining the solution such as time limits, resources and equipment available to the developer and the administration overhead of maintaining data currency after the project is deployed. Environmental requirements will relate to the environment in which the project will be developed and deployed. For example, there is no point creating an amazing virtual reality experience if it will not be suitable for the devices available to the target user. Aesthetic requirements will be determined from organisational style guides and market research about the target audience.

The Australian *Privacy Act 1988* requires that users must give permission before data can be stored about them, and organisations that store data must only use it for reasons the user has agreed to. Organisations must also provide users the opportunity to modify or remove any data collected about them if requested by the user. This is why many websites with online forms provide a privacy policy page explaining what the data will be used for. Although there are international privacy guidelines, privacy laws vary from country to country. Personal data stored on a web server is governed by the specific data privacy laws of the country where the server is hosted, not the country the web page is accessed from.



Constraints are restrictions that could limit the development options for the digital solution. These can relate to any aspect of the project and can include availability or skills of the development team, project budgets and deadlines, ESRB rating guidelines, technical capabilities of the user environment and any other factor that could influence the project direction.

A mind map such as the one in figure 3.2.6 is a great way to explore a digital problem and to brainstorm project constraints and the various functional and non-functional requirements for each of the solution components.

ACTIVITY 3.2.2

Imagine you are developing a digital solution that will do your Maths homework for you. Use a mind map to brainstorm some of the functional requirements, non-functional requirements and constraints for the solution. After 10 minutes, compare your mind map with the one of the person sitting next to you. How long would it take to complete a mind map about requirements for a digital solution as part of a school project? Would it be best to complete the mind map before you start or keep adding to it throughout the project? Discuss your ideas with the rest of the class.



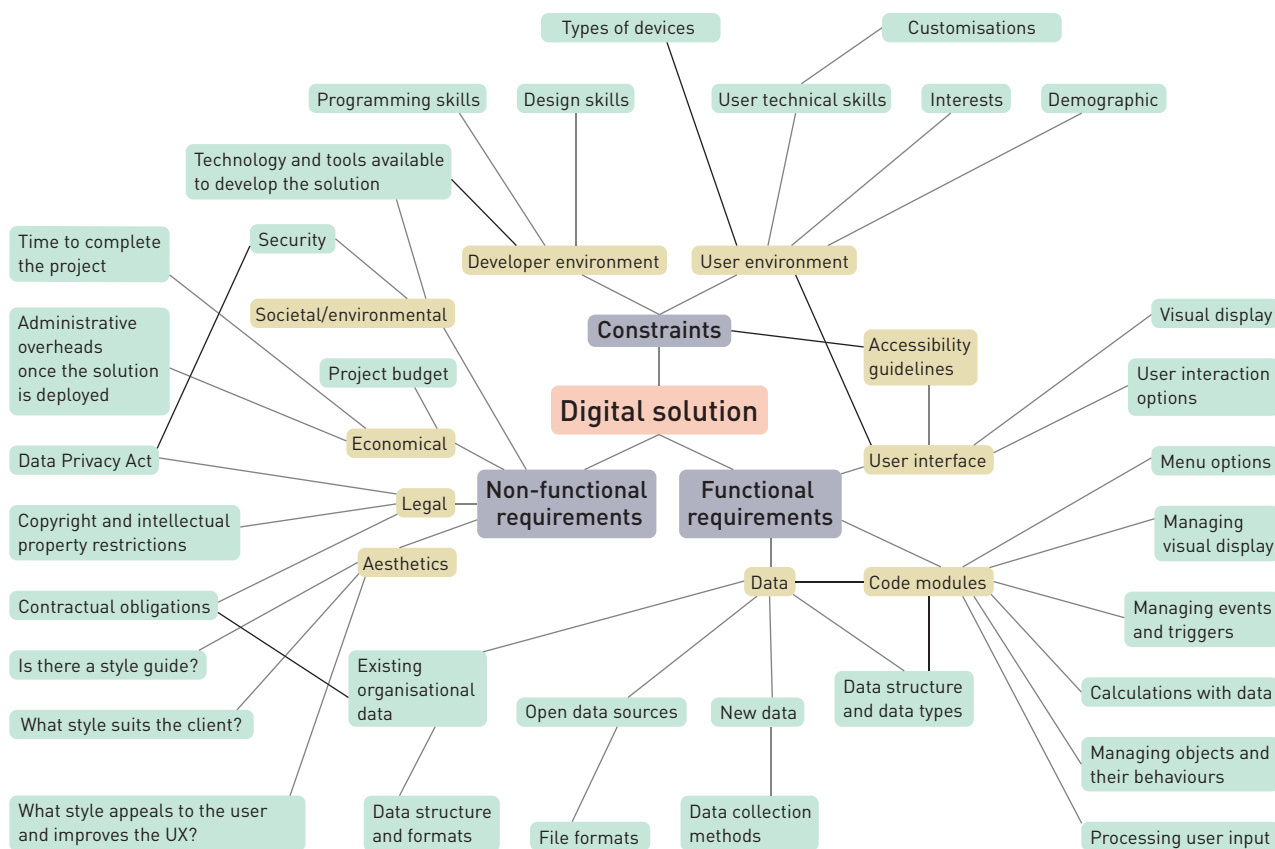


Figure 3.2.6 Mind maps are a great way to brainstorm requirements for a digital solution.

Defining the user interface

Understanding more about how users will interact with your solution will help you cater to the needs of a diverse target audience. Who will be using the solution? When will they be using it? What devices will it need to be compatible with? Are there different groups of users that have different needs? The user groups for an educational tool might be students, teachers and parents. Each of these groups will have different needs and levels of experience. Customers, store managers and central administrators may be the user groups for a sales solution. What features or functions are important to these different groups of users?

To understand the diverse ways in which users will interact with your solution, you can use a number of different techniques to analyse specific target groups. Brainstorming and user profiles, discussed on pages 218–21, are popular ways of understanding more about the interests and behaviours of your target user group. Surveys and interviews are another effective strategy.

Surveys

There are many online survey tools that can be used to set up a list of simple survey questions. Online surveys can be emailed to a wide range of potential users. Many of these tools provide a variety of question options such as multiple-choice, ranking systems and short answers. Some will even graph or summarise the results making it easier to analyse the results. Here are some tips for creating an effective online survey:

1 Introduce the survey.

Explain to people what the survey is for, how long it will take and what will happen to the data. If you are collecting personal information, explain how you will ensure this data is de-identified before the results are published and how you will keep the information secure.

2 Ask only what is necessary.

People have busy lives and are more likely to complete a short (5–10 minute) survey. Try to limit your survey to between five and 10 simple questions. Effective surveys have a specific focus. They include a few multiple-choice questions to collect the most important information and then one open-ended question at the end.

3 Ask a combination of quantitative and qualitative questions.

Quantitative questions are multiple-choice questions or ask for yes/no responses that are easy to analyse in a graph or tally – for example, 65 per cent of respondents agreed/disagreed.

Qualitative questions can give you better information, but are time-consuming to analyse. Some survey tools give an indication of word frequency for qualitative responses, but in most cases a person has to read and summarise each response. Think how much time it will take to review qualitative responses before you add them to your survey.

4 Use audience-appropriate language (avoid technical terms).

Don't assume your audience has the same level of knowledge as you. There is no point asking a user which RGB colour code they would prefer as the background colour for your mobile application because they may not understand this technical language. Instead, list well-known colour names or present a palette of colours to select from.

5 Test out the survey before you send it.

A user requirements survey needs to be presented in a professional way. If you can't write a good survey, the customer may lose confidence in your ability to create a quality solution. Before emailing the survey, check for spelling errors, whether there is enough room for expected responses, and that optional questions are not marked as compulsory.

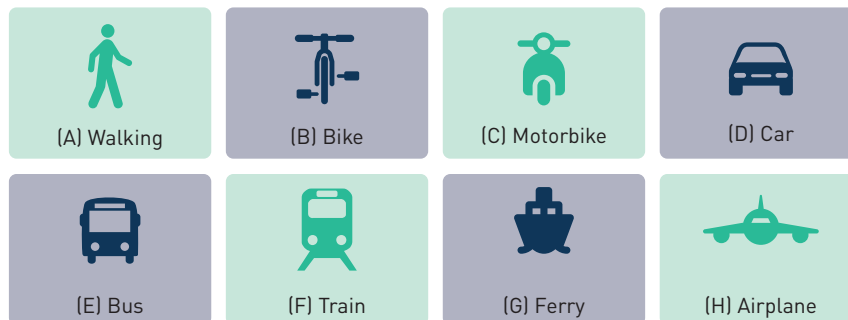
**7 Which form of transport do you use the most?**

Figure 3.2.7 Surveys are a good way to collect qualitative data from a wide range of users.

While online surveys are a quick and efficient way to collect information, it is difficult to authenticate anonymous responses. People may be too busy to complete the survey, or may not answer truthfully.

Interviews

Interviews allow more flexibility than a survey and can be conducted face to face or via a videoconferencing tool (see figure 3.2.8). Notes or recordings of interviews can be a valuable resource that provides a wide range of information helpful in determining the needs of a representative user. Here are some tips for conducting interviews to define the user requirements for a digital solution:

- 1 Select an interviewee who is typical of the target user group.
- 2 Prepare a list of questions ahead of time using the '5Ws and H' method. (The 5Ws and H method is asking Who, What, When, Where, Why and How.)

ACTIVITY 3.2.3

According to *PC Mag* (February 2018), 19 per cent of online surveys are completed on a mobile device. Create an online survey and compare how easy it is to complete on a laptop and a mobile phone. List three tips you would give to someone who is designing a survey for a new mobile phone application.



- 3 Ask the questions using a conversational approach, rather than as an interrogation. The way you ask the questions and the enthusiasm you bring to the interview can make a big difference to the openness of the responses. Treat the person you are interviewing the same way you would treat a friend whom you are trying to help.

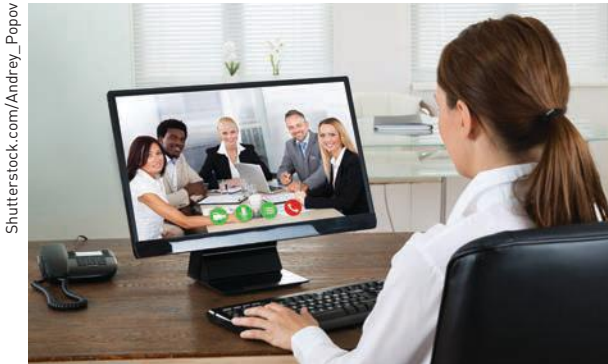


Figure 3.2.8 Video interviews are an effective way to collect user requirements.

- 4 Listen to the responses carefully. Don't move on to the next question until you have all the information you need from the current question. Summarise what you understand and repeat it back to the interviewee to show that you have listened to them, and check that your understanding is correct.
- 5 Take notes or record the interview. One of the advantages of videoconferencing is that it is usually possible to record the call using features of the videoconferencing software. Don't record the interview unless the interviewee agrees beforehand to be recorded. If recording is not possible, take notes during the interview.



Explore

GROUP ACTIVITY 3.2.4

- 1 Practise the interviewing tips above by conducting a mock interview with a classmate. Use the list of questions below to explore how your classmate would feel about receiving their fast food order from a robot.
 - **Where** is your favourite place to order fast food?
 - **When** was the last time you ordered fast food? Time of day? Day of week? Snack time? Meal time?
 - Thinking about the last time you ordered fast food, **who** took your order and/or served your food? What did you like/dislike about the way that person communicated with you? How is it better or worse than if a robot served your food?
 - **What** did you order? Would it be difficult to order those items from a robot? How would that be a better/worse experience than ordering from a human?
 - Do you think fast food companies will employ robots in the future? **Why/why** not?
 - Describe **how** a robot might take your order and serve your food. How would that change your fast food experience?
- 2 Summarise your notes from the interview and identify at least three requirements for a fast food robot.
- 3 Reverse roles with your classmate so they can interview you and determine their own list of requirements.
- 4 Compare your list of requirements with the list created by your classmate and look for similarities and differences. Work with your classmate to combine the two lists and create a single list of four requirements or features. Place the requirements in order from most important to least important.
- 5 Reflect on your interviewing techniques and make a note of what you did well and what you could improve next time.

Defining the data requirements

In this unit, all the digital solutions you create will involve some type of data storage. The data may be stored in files containing individual data values or an interactive database that allows selection from tables of data. Some of the data will already exist and can be downloaded and integrated into your solution. Some new data structures may need to be created. Defining the data requirements will be an iterative process that you may revisit several times. It is not always possible to identify all the data you need until you have created

a few solution prototypes. Here are some tips to help you get started with defining the data requirements for your digital solution:

- 1 Analyse the user problem to identify all the types of data that might be needed. This can be done through brainstorming, concept maps or rough sketches and diagrams.
- 2 Sort this data into data that exists already and data that needs to be created.
- 3 For each data item that exists, explore the type and format of the data. For example, is the data stored in a CSV file, as an XML file on a website, in mobile phone local storage or in a database? What fields are available in the file and what is the length, data type and example value for each field?
- 4 For data that needs to be generated, identify whether the data will be provided by user inputs or stored records. Describe the type of data that will be generated and what format it will be stored in.
- 5 Investigate and compare the type of data structures available for your digital technologies context. What types of data tools are available in the school environment? What formats are required by these tools?
- 6 For data that exists, identify any format changes that might be required before the data can be used in your solution.
- 7 For data that needs to be generated, build a dictionary of data definitions explaining the format for each data item and how it will be used in the solution (e.g. length, data type and example value). Review the data definition process from Unit 2 to define the structure of new data that will be created for your solution.

There is no standard format for a data definition. Use brainstorming, concept maps, tables, textual descriptions, and free-form sketches to define the initial data requirements for your solution. Unit 2 explains how to represent data requirements using a data flow diagram. Figure 3.2.9 shows how a data flow diagram can be used to symbolise the relationships between different data stores used by a digital solution.

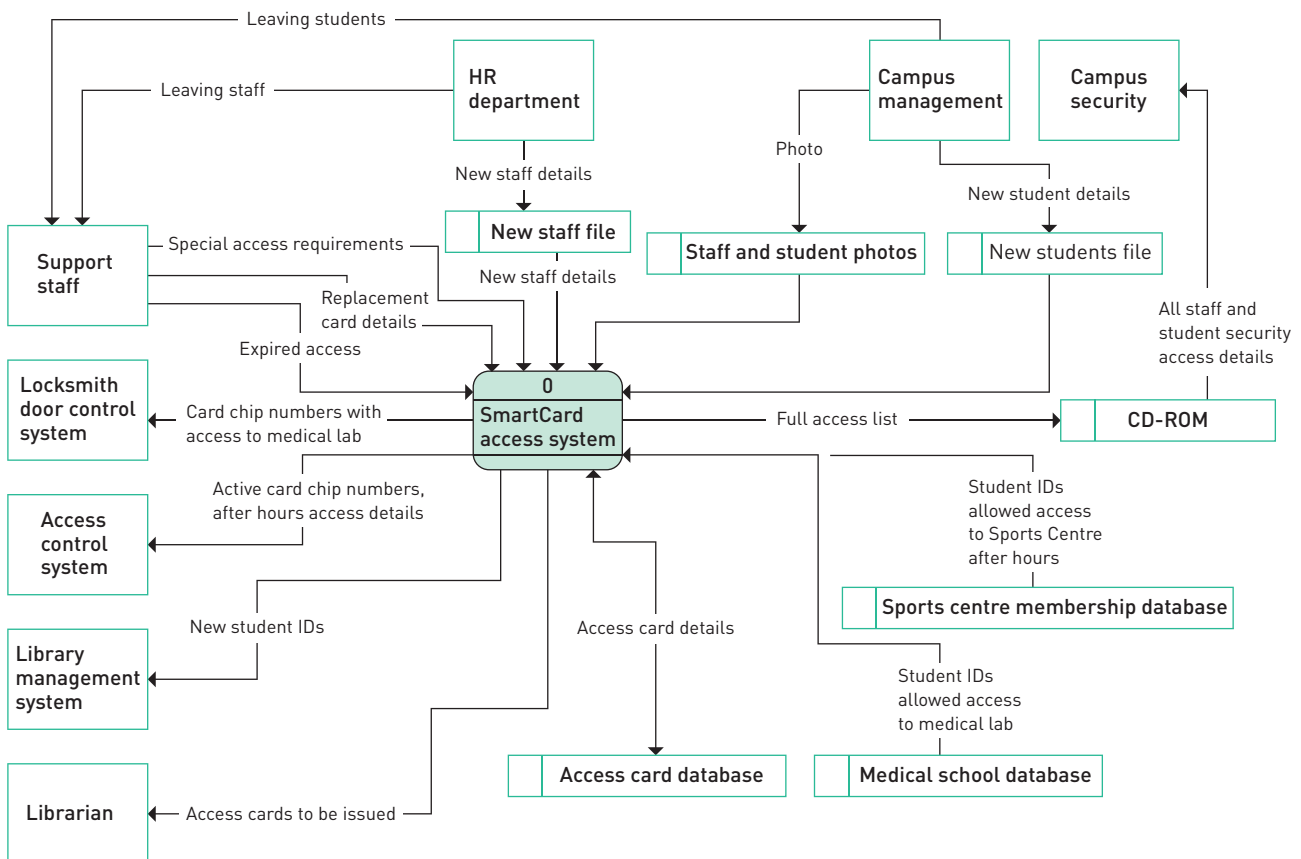


Figure 3.2.9 This is an example of a data flow diagram for the Newcastle University SmartCard system. It shows a number of different types of data inputs and outputs including CD-ROMs, databases, batch files and manual user input.



ACTIVITY 3.2.4

Explore the diagram shown in figure 3.2.9 to explain the types of user interactions and data stores it symbolises.

- 1 **a** People and systems are different types of entities, but both are represented as a square in the diagram. How many different systems request or receive data from the SmartCard access system?
- b** Which users request data from the system?
- 2 Which user(s) enter data directly into Newcastle University's SmartCard system?
- 3 How are data stores represented in the diagram?
- 4 Where would a new student need to go to collect their SmartCard?
- 5 What types of students have special access requirements?
- 6 What data flows might be used if a staff member or a student misplaced their SmartCard?

Defining the coded components

The coded components are the 'brains' of your digital solution. They control the logical sequence of events that should occur for each interaction between users and data. Some program modules may already exist in code libraries and some program modules may need to be developed or modified. Requirements for the coded components will evolve throughout the development process. The initial requirements may be a high-level list of menu options, data-processing statements or actions that occur in response to user input. The requirements will become more specific as the project evolves and will be refined regularly as part of the evaluation phase of the problem-solving process.

The requirements for coded components will vary greatly depending on the technology context selected. It may be possible to brainstorm a list of functions required, but until the programming environment or digital technology context is selected, it will not be possible to create a detailed set of requirements for these components.

In a web application, the coded components will likely be embedded in or linked to a web page. These modules are likely to be triggered by HTML form elements. They could be functions that validate a web form or format data. They could be a coded web page that establishes a link to, and exchanges data with, an online database server. It is possible that a number of different programming languages will be used, such as JavaScript, PHP and SQL. An annotated site map such as the one in figure 3.2.10 is a good way to define the requirements for the coded components of a web application.

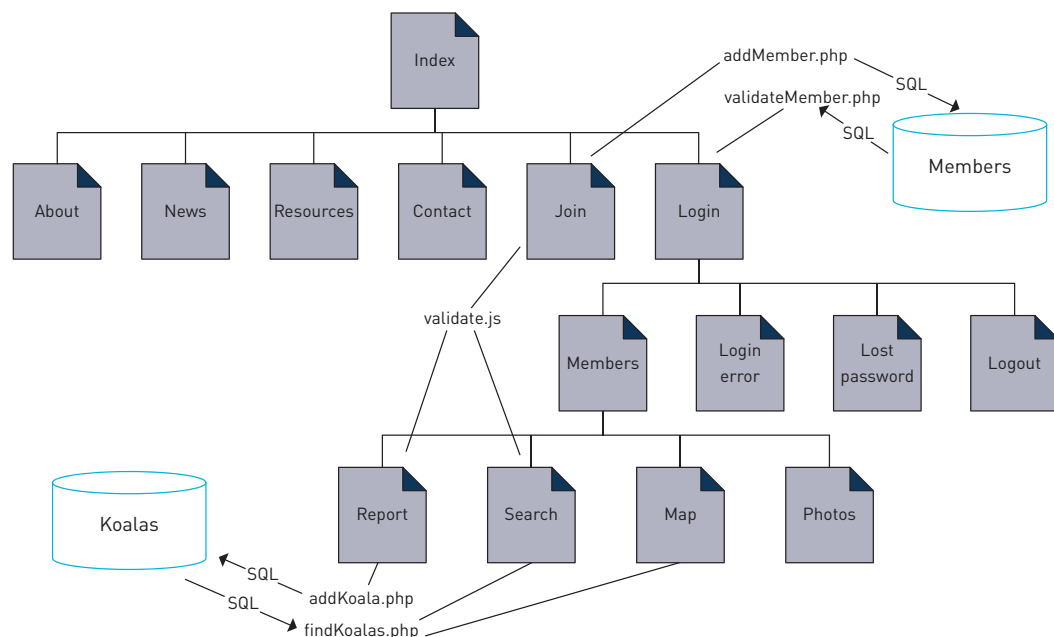
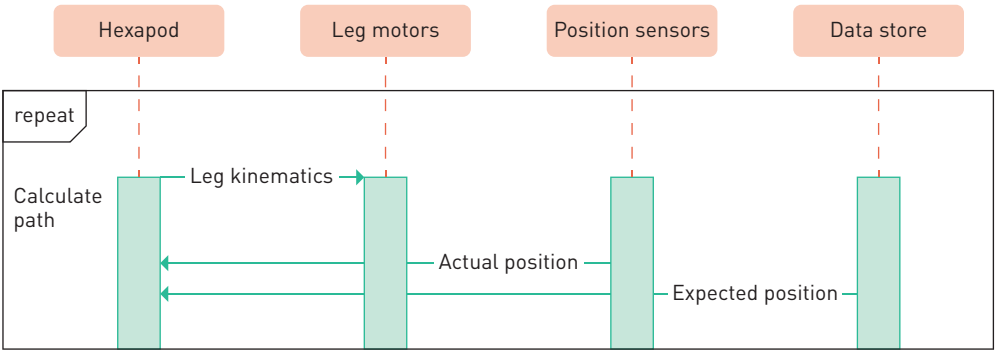


Figure 3.2.10 An annotated site map can be used to identify the required components of a web application.



An intelligent systems environment will have coded components that transmit instructions to actuators in response to sensor input. Some code will control the actions of a single component of the solution, while other components will control the overall sequencing and operation of the solution and may need to establish synchronisation points to coordinate the actions of multiple program components. An interaction sequence diagram such as the one in figure 3.2.11 is a popular way to represent the requirements for the complex interactions that occur in intelligent systems.



3.2.8 UML 2
sequence
diagrams
3.2.9 UML
sequence
diagrams tutorial
3.2.10 UML class
diagram tutorial
3.2.11 UML class
diagrams tutorial,
step by step

Figure 3.2.11 An interaction sequence diagram can be used to show connections between components in an intelligent system.

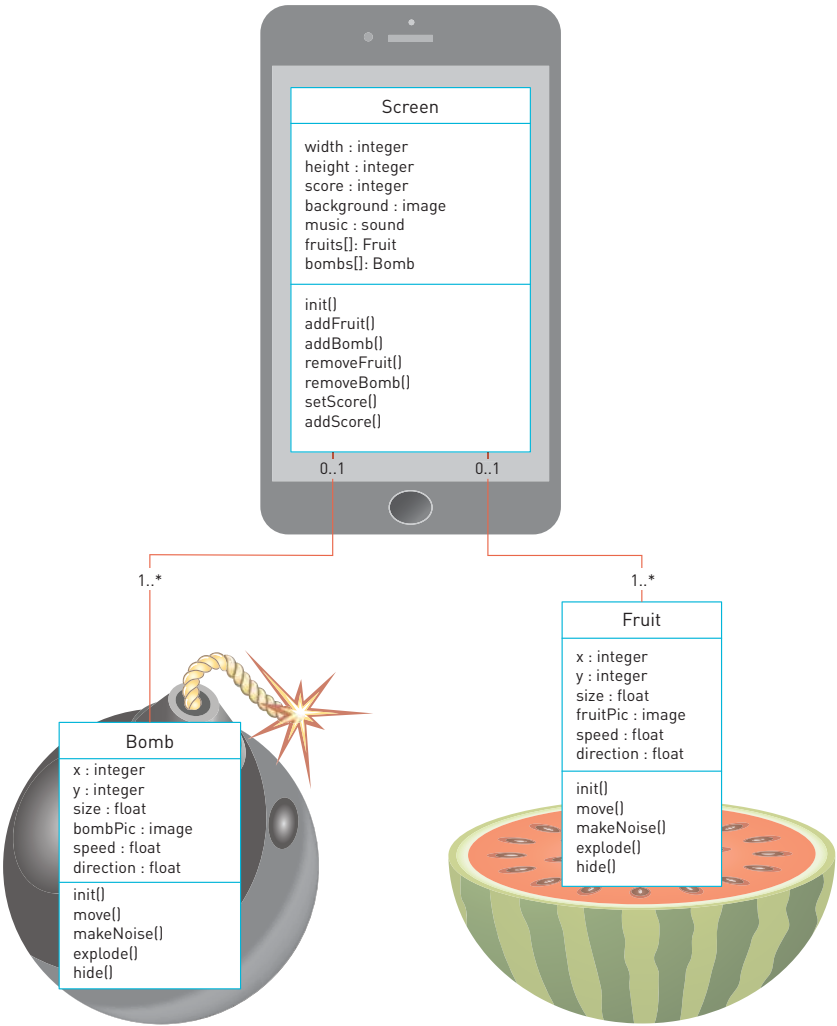


Figure 3.2.12 This UML diagram identifies the object properties and methods for a mobile game like Fruit Ninja.



Many interactive media and mobile applications use object-oriented programming languages. In these technology contexts, the coded components will usually be defined as a list of objects and the methods associated with each object. A Unified Modelling Language (UML) diagram (see figure 3.2.12) is an easy way to identify the objects and methods in an interactive media or mobile phone application.

Diagrams are a great way to symbolise requirements because they can communicate a lot of information clearly and succinctly. Annotations or correct use of formal symbols can explain in more detail how components of a solution will interact. It is also important to describe requirements as a written checklist that can be used to evaluate the completeness and success of each component of the solution.

Evaluating the requirements

The prescribed criteria identified at the commencement of a project will essentially become the checklist of success criteria that the solution will be evaluated against. Developers and clients can usually imagine more features than can be completed in a short timeframe, so scoping the solution is important to manage the expectations of what can be delivered before the project deadline. Often, software will be released with basic features that are extended in later versions. This helps to manage the workload of the development team and set reasonable client expectations. After each brainstorming session, the list of possible features increases, so it is important to evaluate the list of requirements to separate the needs or ‘must-have’ features and the wants or ‘nice-to-have’ features. Requirements can also be evaluated with a time or difficulty level factor to assist with planning and time management of the solution. There are many approaches to prioritising requirements. The user story in figure 3.3.2 (page 252) includes a priority column to help the development team focus on the most important features. Another approach, referred to as the MoSCoW method (Must, Should, Could, Won’t), is illustrated in table 3.2.1.

Table 3.2.1 The MoSCoW method helps prioritise solution requirements for games like Fruit Ninja.

MUST	SHOULD	COULD	WON'T
Be simple enough for your mum to play	Have fruit that the users can ‘smell’	Have different modes	Be free
Be suitable for mobile devices	Have juicy bright coloured fruit that explodes	Have a featured character	Refer a friend
Use the touchscreen	Have randomness	Be multiplayer	Show advertising
Increase company revenue	Use swipe as the primary user interface	Have multiple blade options	Have tomatoes
	Have bombs to prevent players from cheating	Link to GameCentre	Have vegetables
	Have a score	Have a Kinect version	
	Have a swipe tutorial		

Developing the user interface

The user interface is the way that a user interacts with the digital solution. In most digital solutions, this will be a graphical display, but it will depend on the technology context. Many user interfaces for intelligent systems are physical interactions such as biometric sensors, buttons or switches. In some cases the interaction is triggered by another program or a machine. Regardless of the type of end user, it is important to ensure that the user interface is designed in a way that makes it easy for a target user to interact with the digital solution.

Useability

Early user interfaces were developed to achieve maximum efficiency for computer hardware. Only expert users were able to successfully interact with computers, and they were expected to learn hardware-specific coding languages in order to obtain the required output from the computer. As computers have become more powerful and a wider range of users are required to operate computers, software has become more user-friendly. **Useability** is a user-centric set of principles that aim to improve the user experience by making computers easier to use. The useability principles are:

- accessibility
- effectiveness
- safety
- utility
- learnability.

Without useability a digital solution is not useful. It is important to plan for useability at the start of a project and evaluate the useability of a digital solution through user feedback before releasing it for general use.

Accessibility

Computers have become an essential tool for all citizens, and in many situations – for example, with schools, government services and commercial enterprises – digital interactions have completely replaced, or are in the process of replacing, traditional forms of communication. In many organisations, digital forms have completely replaced paper forms. However, not all students or adults have access to a computer, and even some who do are not able to experience the digital world in the same way as the majority of users. Many physical, cognitive, auditory and visual impairments have the potential to disrupt the digital experience.

Ask a child with cerebral palsy if they have ever played a video game, and you will likely find out that they have never had that opportunity because they lack the coordination to interact with a standard games controller. How does someone who is unable to move their hands log in to a computer? How does someone who cannot see a computer screen complete a search or book a movie ticket online?

Accessibility guidelines provide organisations with recommendations for how to provide an equitable digital experience for marginalised users who rely on assistive technologies to interact in our increasingly digital world. In a web application, accessibility can improve the way screen readers describe web pages for the blind. In desktop applications, it can ensure that the tab key order is logical and will support interaction from a keyboard controlled by the chin. In interactive media, it can recommend configurable options for games controllers to allow custom-made devices to be used, and in mobile applications it can recommend configuration options that reduce cognitive load by allowing sounds, flashing alerts and notifications to be deactivated and ensure that colours and button sizes are appropriate.

Regardless of the technology context, accessibility is an important consideration in the development of a digital solution. Failing to consider accessibility during the development of a digital solution is considered a form of discrimination in many countries. In Australia, the *Disability Discrimination Act 1992* requires all government agencies to ensure that all information and services, including web applications, are accessible to all Australians.

'We need to make every single thing accessible to every single person with a disability.'

Stevie Wonder, 2016 Grammy Awards

ACTIVITY 3.2.5

Create a list of 10 accessibility considerations that are relevant to one of the digital technology contexts described on page xiv.



3.2.12 IGDA
game access
SIG

3.2.13 Mobile
applications
and litigation



Explore



3.2.14 Web
accessibility
perspectives



Figure 3.2.13 Guru of useability, Jakob Nielsen



3.2.15
10 Usability
heuristics
for user
interface
design

According to the *New York Times*, the guru of useability is Jakob Nielsen, pictured in figure 3.2.13. Nielsen developed 10 rules of thumb for improving useability in 1994, and these 10 useability heuristics are still widely used two decades later. See page 279 for more information about using useability heuristics to evaluate your user interface. Nielsen's useability heuristics have been published and referenced in many textbooks, online blogs and YouTube clips.



Effectiveness

An effective user interface will be one that reduces the possibility of user error. This can be achieved by restricting input options through the use of drop-down boxes or input templates, and by providing simple, clear instructions with a limited number of steps. Simplicity and consistency are key elements of an effective user interface.

Safety

In 1942, long before robotics became a reality, author Isaac Asimov penned the 'First Law of Robotics' in his novel *I, Robot*. It states that robots should never harm humans. Several decades later, robotic co-workers are no longer fictional and this 'law' is now quoted in scientific discussions about robotic ethics. The concept of human safety also applies to computers. Shouldn't a computer also 'do no harm' to humans or to the work they have created? 'Undo' buttons, warning messages and autosave features are all examples of safety features built into essential software. Safety is the useability principle that gives the user the option to change their mind and reduces the likelihood of catastrophic errors. Without safety features, humans would find it very difficult to trust computers, and digital solutions would not be considered reliable.

Utility

Utility involves providing functions the user needs. It can also ensure that the functions the user needs most often are most prominent. Understanding how and why a user will interact with a digital solution will help identify important features that must be included and the best placement for frequently accessed options and buttons. It is important to

evaluate the utility of a digital solution to ensure that it fulfils the minimum requirements. For example, a shopping website with good utility will ensure that the user receives a receipt for each transaction. A mobile game application will have some type of feedback that shows if the user has won the game.

Learnability

Learnability is how easy it is for users to interact with your solution. One way to provide learnability is to use real-world representations that the user is already familiar with. Another way is through 'help' icons or tutorials. The importance of learnability is evident from the experience of Halfbrick Studios, a Brisbane-based game development company. During the useability testing for its *Fruit Ninja* mobile game, many users handed the game back to developers, saying it didn't work. The users selected for testing were random strangers, waiting for the bus, who had never played a mobile game app before. These novice users weren't used to slicing buttons instead of tapping them, so gave up quickly. The user experience (UX) designer resolved the problem with a simple animation, triggered after a few taps on the button, showing that it was necessary to slice the buttons the same way the fruit needed to be sliced in gameplay (see figure 3.2.14).



Figure 3.2.14 Learnability can be achieved with a simple 'help' animation such as the swipe animation in the Fruit Ninja game.

ACTIVITY 3.2.6

Reflect on the first time you used one of the software programs at your school. What helped you to become familiar with this software? Evaluate the learnability of one of the applications installed on a computer, laptop or tablet device at your school. Categorise software features as easy or difficult to use.



Responsive interfaces

Responsivity is the degree to which something responds to external influences. In the context of usability, this is commonly understood to be the ability of a programmed user interface to adjust to the hardware display device. This is a very important consideration when it is possible that users will be accessing your solution from a variety of different sized screens. Web applications in particular can be accessed from a variety of user devices, including desktop displays, tablet devices and mobile phones. Designing for the small screen of a mobile phone, or a tiny LED display used by a robot or microprocessor, can also be a challenge. It is extremely frustrating for users attempting to access a web page on a mobile device if the layout has been designed for a much larger screen, making buttons too tiny to be practical.

Low-fidelity prototypes

Low-fidelity prototypes are throwaway sketches or rough digital mock-ups that can be used in early walk-throughs or UX testing. They are used for planning purposes only, and allow the developer to quickly compare and test different ideas. Sketches, mood boards, storyboards, site maps, wireframes and mock-ups are all examples of low-fidelity user interface prototypes.

Paper walk-throughs such as the one shown in figure 3.2.15 are a good example of low-fidelity prototypes. They are a fast, low-cost method of testing design decisions for the user interface and they facilitate observations about how users will navigate through the different options presented. The designer represents each interface or screen on a single piece of paper to simulate the final product, and potential users interact with the paper prototype by pressing the paper buttons. In response to the paper button press, the designer swaps the paper with the next screen that would appear or describes what would happen. This can be very useful for identifying frequently used functions, the number of clicks required to complete a task, and even missing buttons or screens. Research has shown that paper walk-throughs with five real users are sufficient to find 80 per cent of potential problems.



Figure 3.2.15 Paper prototype for a mobile phone application



ACTIVITY 3.2.7

Practise the technique of paper walk-throughs with an existing web application or mobile phone app you are familiar with.

- 1 Cut out a few pieces of blank paper that match the dimensions of the device you would typically use to access the application.
- 2 Explore one aspect of the application, observing the different screens, drop-downs and pop-ups that are presented.
- 3 Sketch the main templates of the screen onto the blank paper using a pen or pencil. For example, if it is a website, sketch just the banner, footer and navigation bar. Remember that a paper prototype is supposed to be a quick, rough sketch. Do not spend a long time trying to make it perfect. Focus on a black-and-white outline of the important components.
- 4 Create smaller cut-outs to represent drop-down lists, pop-up messages and content sections that can be added to each page. Post-it notes work well for this as they are easy to stick onto the main template.
- 5 When your paper prototype is finished, test it out with a friend. Ask them to use the paper prototype to access the aspect of the application you have re-created. During the walk-through, focus on observing the user interactions, and talk as little as possible. Afterwards, encourage your friend to share their impressions of the walk-through experience, including what they liked and anything that was frustrating or confusing.
- 6 Think of a new feature you could add to the existing application and modify your paper prototype to include it.
- 7 Complete a paper walk-through with a potential user and observe how they interact with the new feature. If you have time, conduct the walk-through test with up to five different users.
- 8 What did you learn from this activity? What did you find surprising?

Elements and principles of visual communication

Studies conducted by the Interaction Design Foundation have shown that good aesthetics is an important feature of the user experience. If users enjoy the aesthetics of the product they are interacting with, they are more forgiving of usability issues. Applying the elements and principles of visual communication to user interface design improves the user experience by engaging the emotions of the user and increasing motivation to revisit the experience or complete a difficult task.

As discussed in section 2.1, the elements of visual communication are space, line, colour, shape, texture, tone, form, proportion and scale, and the principles are balance, contrast, proximity, harmony, alignment, repetition and hierarchy. Table 3.2.2 describes some of the ways these elements and principles can be applied to impact the aesthetics of a digital solution, and hence the user experience.

Table 3.2.2 Elements and principles of visual communication

ELEMENTS		PRINCIPLES	
SPACE	Space is the area you have to work with. It includes space that is empty and space that contains text, images and other designed features. The use of space can create clutter or simplicity.	BALANCE	Balance occurs when the combination of elements on one side of a design appears equal to the other side. The weight of the design on each side can be influenced by use of colour, space and shape. For example, adding a dark colour to a small empty space on one side can balance a large area of black type on a white background.
LINE	Lines can direct the user to a focal point, divide different sections, form shapes and provide emphasis. They can be thick, thin, straight or wavy. They can create order or encourage creativity.	CONTRAST	Contrast is used to emphasise an element of the design by using a markedly different colour, size or other characteristic. Contrast is also an important factor for readability of text. It is important that there is high contrast between the colour used for typeface and its background colour.
COLOUR	Colour can provide contrast or emphasis, unify a design, fill a shape or space and imply emotion. A colour scheme can be inspired by a logo or photograph. Colour theory defines rules that can be used to select aesthetically pleasing colour combinations.	PROXIMITY	Proximity is the positioning of elements near other elements. Proximity is an important factor in useability because it is easier to interact with a digital solution if similar functions are in close proximity to each other, such as the navigation links in a webpage.
SHAPE	Shapes can be natural, abstract or geometrical. They can be used to symbolically represent an action or create proximity by grouping similar elements.	HARMONY	Harmony is the pleasing effect of combining similar elements such as using similar colours, typeface and shapes throughout the design.
TEXTURE	Texture can be used to add interest to a shape or space, attract attention or unify a design by linking it to a real world idea with a similar texture.	ALIGNMENT	Alignment can be used to provide unity and proximity to the different elements used in the design. Good alignment aids readability and shows connections between different elements. Objects that are perfectly aligned provide a cleaner aesthetic and more professional appearance which inspires trust and confidence.
TONE	Tone is the lightness or darkness of a colour and can be contrasted to provide hierarchy, emphasis or separation of content.		
FORM	Form is the combination of other elements to create solid effects such as the use of tone and shape to create shadows or the use of line, shape and colour to create a 3D perspective.	REPETITION	Repetition of the same element throughout the design brings consistency, unity and cohesion. For example, repetition of the same icon or button shape can reduce cognitive load and improve learnability for the user. However, too much repetition can become monotonous. Small variations in line, tone or colour to a repeated element can restore interest in the design and helps to retain user attention.
PROPORTION	Proportion relates to the size or quantity of elements in the design in comparison with other elements. Images with disproportionate height and width appear distorted. Unbalanced proportions are usually unattractive but can be used strategically to provide emphasis.		
SCALE	Scale refers to the size of an element. It can be used consistently to provide order and repetition to a design or contrasted to create a focal point or emphasise the importance of one element over another.	HIERARCHY	Hierarchy improves readability of the interface by emphasising more important elements with placement or size. Larger elements or those positioned towards the top of the design are considered more important.

Applying elements and principles of visual communication

When developing ideas for the user interface, it helps to analyse the aesthetics of similar types of digital solutions to understand what makes them effective. The annotations in figure 3.2.16 identify how the elements and principles of visual communication can be used to analyse a website design.

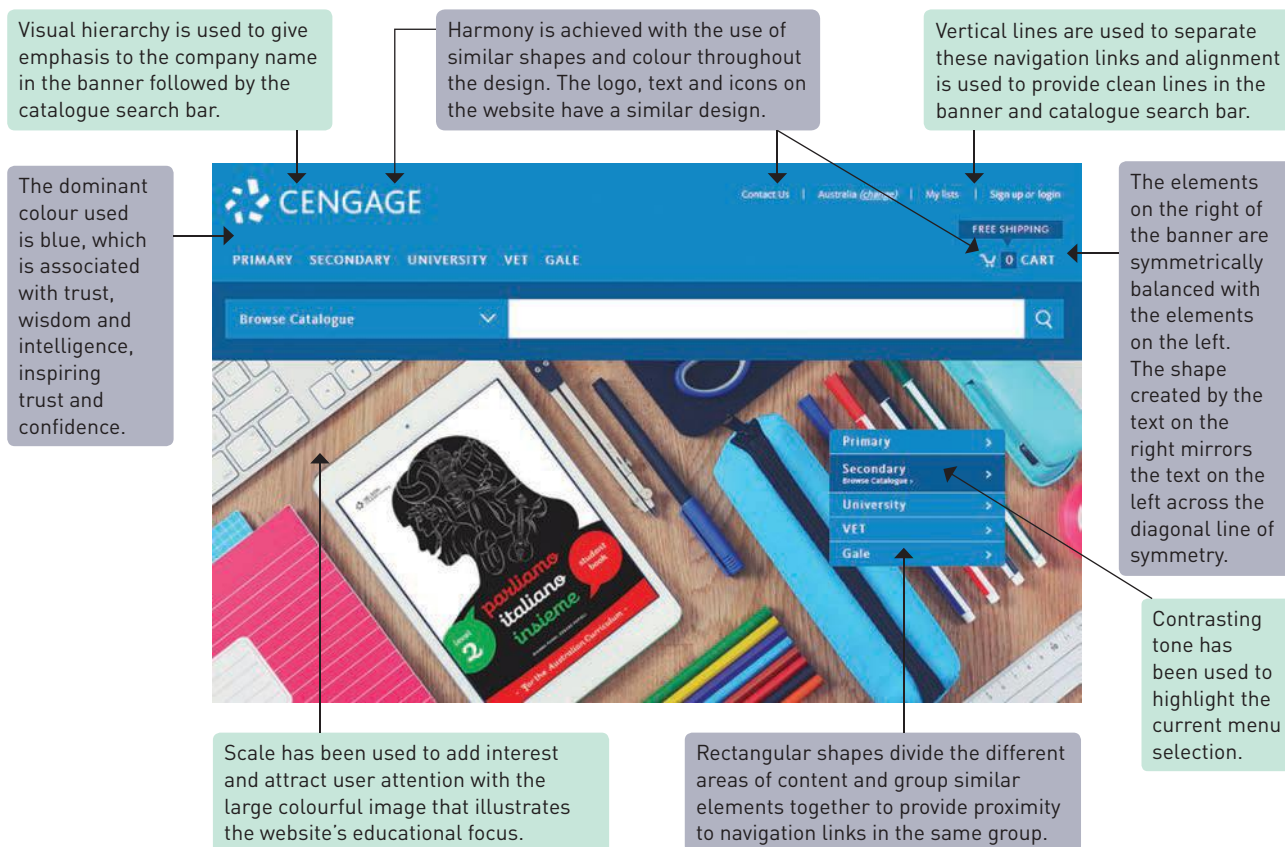


Figure 3.2.16 Elements and principles of visual communication can be used to analyse the design of a web page.

ACTIVITY 3.2.8

Take a screenshot of your favourite web page and annotate it to identify the elements and principles of visual communication that have been used.



Evaluate

Developing the data components

Data is a critical component of many digital solutions. There are many different formats and data structures that can be used to store data. Sometimes an established data source exists and needs to be copied or merged with other data structures. Sometimes new data sources need to be defined and developed.

Common data formats

The two most common types of data formats used with digital solutions are flat files and relational databases.

A flat file is a sequential list of data that must be read and processed in order from the first line to the last. The bigger the file, the longer it takes to read it. To access the 575th line in the file it is first necessary to read and discard the first 574 lines. However, adding data to the file is very efficient. New data can be quickly and efficiently appended to the end of the file. Flat files are often used for logging data that will not be accessed very often, such as system logs and audit trails. Due to their simplicity, they are also a common format for

sharing data in a single file accessible to a wide range of different systems. A flat file can only be accessed by one program at a time, so it is not suitable for data that needs to be accessed by multiple users at once. Data files can be secured using encryption or file level access controls.

A relational database is a combination of indexed tables linked together with unique keys. Accessing data from a relational database is very efficient due to the index structures stored in memory, so any specific data item can be accessed independently, provided the data key is known. Adding new data is not as efficient as with a flat file as there are many constraints and links to maintain, but it is still reasonably efficient and the constraints and links ensure that the accuracy and integrity of data in a well-designed relational database is of a high quality. Most relational databases can only be accessed with a system-specific relational database management system, so exporting data to a set of flat files is usually required when the data needs to be shared with other systems. However, a database management system allows multiple programs to access the data at once, so it is a good format for a client server application where many programs of the same system type need to share data. Database management systems are generally more secure than flat file systems and can be configured with complex levels of security, controlling who can read or update specific data items.

Selecting an appropriate data format

The best type of data format to use in a given digital solution will depend on the data requirements.



Figure 3.2.17 Questions to ask when selecting a data format

Explore the following scenarios to consider how data requirements and the technology context can influence the choice of data format for a specific digital solution.

Selecting a suitable data format for a robotics solution

Databases are important for robots to store persistent information about their operational state, environmental data such as maps and rules, and task status information. Robots also use flat files to store data collected from sensors and for time-based logs and activity data.



Consider a robotic scanner. The robot scans an item barcode. It then looks up the price of the item and displays it on an LED display. The robot continues scanning items and calculating the total of all items. If it encounters an item for which it can't find a barcode, it beeps and displays the message, 'Please scan again'. During its operation the robot keeps track of the time taken to scan each item and whether the scan was successful or not. This operational data is later analysed by the developer to recommend improvements to the robot scanner program.

Data requirements for the robotic scanner include the following:

- Retrieving barcode data is time-critical and needs to occur in the shortest possible time.
- Barcode data is frequently read but rarely inserted or updated.
- Scan duration and success are frequently written and rarely read.
- Barcode data contains many records – possibly too many to store in the robot memory.
- Scan duration and success can be downloaded and archived regularly.
- The data is not sensitive and does not need to be protected.

Based on these requirements, operational data about the scan duration and success would be best stored as a flat file on the robotic device and archived daily for later analysis by the developer. Barcode data contains many records that need to be accessed quickly, so a relational database would be the best data format for the barcode data.

Selecting a suitable data format for a web application

Web applications can store data on the local client machine or on the central web server. User-specific data such as session information is stored on the user device, and system data relevant to all users is stored on the web server.

Consider a shopping website. The price of products available for sale will be accessed regularly by visitors to the site. The general list of products will rarely change, but the price of sale items could change weekly or monthly. When visitors to the site make a purchase, they will need to enter personal information about who to address the delivery to and sensitive information about payment details. The information about delivery details will need to be stored reliably until the delivery is completed, but payment details are not stored at all because the customer is transferred to a secure payment gateway system that processes the payment on behalf of the shopping website.

Data requirements for the shopping website include:

- Information about products needs to be accessed by multiple users at once.
- Access to information about products is time-critical as users will leave the site if it takes too long to retrieve the product details.
- A large number of products and their related product details need to be stored.
- Personal information about customer purchases needs to be kept secure.
- Purchase information can be provided by multiple customers at once and needs to be accessed at least daily by sales staff at a central location so that orders can be filled.
- Payment information will be managed by an external system, so only the payment method and whether the payment was successful needs to be recorded.

Based on these requirements, a database would be the best file format to store products and prices because these data items are accessed regularly. Due to the need for concurrent updates, customer order details, delivery information and payment status should also be stored in a database. Access controls for the database should allow all users to read products and prices but allow only the shopping website managers to update prices. Personal details about customer purchases should be kept private from other customers, but sales staff should be able to read information about orders that need to be filled. It is possible that session data could be recorded in a flat file or in local memory on the user device.

Selecting a suitable data format for a mobile application

In the past, mobile application databases have been stored remotely on an Internet-accessible server or distributed throughout several database servers at various geographical locations. Accessing remote data from

a mobile phone relies on the speed of the mobile network connection and is often much slower than users would like. More recently, the local storage capacity of mobile devices has increased to the point where it is now feasible to embed a database management system such as SQLite within a mobile application stored on the mobile device. This allows efficient access to locally stored data. If required, a local copy of the database can be synchronised with a copy on the main server, ensuring the data is available centrally without impacting the speed of the user experience. Mobile phones also store data in flat files.

Consider a mobile application that records your tax receipts.

When a user receives a tax receipt, they record the details in their mobile app. Most of the time the application will store the data locally, but at tax time the user can choose to upload their tax records to an online tax return. Most of the data stored on the mobile phone is sensitive financial data requiring a high level of security. The data can be archived or deleted each year after the submission of the tax return, but original copies of the receipts will need to be kept for the time specified by the taxation office.

Data requirements for the tax receipts mobile app include the following:

- All data stored on the app is sensitive and requires a high level of security.
- Data is frequently added but only occasionally accessed.
- Data is only accessed by one user and occasionally shared with a central server.
- It is unlikely that a large number of receipts will be recorded in the app.

Based on these data requirements, a flat file format would be suitable for this mobile application but the file would need to be encrypted to protect sensitive data. A relational database would also be suitable, and may improve the efficiency of updating or correcting tax receipt data.

Selecting a suitable data format for an interactive media solution

Some interactive media solutions, such as games and virtual reality simulations, require fast response times during gameplay. Many game developers believe it is not effective to incorporate regular updates to a flat file or database. However, storing some game data, such as player profile details, high scores and game progress, can improve the user experience. In many cases a game-specific flat file is used to store persistent data about game progress during user-initiated save points, but many complex multiplayer adventure games with detailed storylines, character customisation and inventories, such as World of Warcraft, use a relational database.

Consider an online multiplayer game of strategy that allows you to interact with other players in a chat room and form alliances. The chat room shows which players are active and which players are offline, so, when a player logs in, their login status needs to be shared with the other players in their alliance. The alliance will have a leader that can change from time to time and specific bonuses that are shared with the members of the alliance.

The data requirements for this game are:

- Alliance details are occasionally updated and regularly accessed by alliance members.
- Player profile data is occasionally updated and regularly accessed during gameplay.
- Some of a player's profile data, such as login details, is private, but other details such as their alliance membership and login status are shared with other players

Based on these requirements, a relational database would be the best choice to store data in this game as it can be accessed concurrently by all the users in the alliance and will be the fastest way to retrieve and update shared player and alliance data. A relational database will also allow different levels of security to be configured.

Combining data sources

It is not always necessary to create new data for every digital solution. In some cases, the data source for a digital solution is created from, or merged with, a subset of existing enterprise or open data. When creating and merging different sources of data, it is important at all times to maintain data accuracy and integrity. If the data will need to be updated over time, it may be necessary to develop automated processes to sustain the currency of the data.

Cleaning data

Data created for one purpose can sometimes require adjustment before it can be used for a secondary purpose. Cleaning data is the process of updating a data source to remove irrelevant details, incompatible formats or inaccuracies, and replacing or modifying missing data.

In order to develop a data-cleaning strategy, it is important to understand the weaknesses in the data-collection system. If the data was collected through online surveys, it is possible that some form fields have been left empty or contain spelling errors and false information. If the data has been collected from electronic sensors, duplicate entries or out-of-range data could indicate inaccuracies or corrupted data and should be investigated further to determine if they should be excluded from the dataset. Depending on the size of the dataset, some of the cleaning can be done by eyeballing the data and looking for anomalies or atypical patterns in the data. Cleaning can be automated by writing programs that read each line of a flat file and test for and correct common error patterns, or updating selected rows of a relational database with a single SQL statement.

Merging data sources

Flat file data sources can be merged by reading data from each file and writing it to a new file using an automated program. Tables in two different relational database systems can be merged by exporting the table from one system and importing it into the other. A relational database can be merged with a flat file by importing a flat file into the relational database as a new table structure.

Before merging data sources, it is important to:

- remove duplicate data
- identify key fields
- establish **domain integrity**.

For example, a car registration number is used as a key field to record data about car ownership in the Department of Transport and Main Roads vehicle registration system, and is also used to record parking fines issued by the local council authority. Can these data sources be merged using the car registration key field to find an address to send the parking fines to? In most cases this would correctly identify the address of the driver who incurred the parking fines, but it may be possible that the current registered owner purchased the car second-hand and that the parking fines were incurred by the previous owner. The duplicate vehicle registrations after the date and time of the ticket need to be excluded before merging the datasets, and only the most recent vehicle registration in the remaining data should be used.

Sometimes the domain integrity of the key field also needs to be considered. When linking information by the key field 'state', an abbreviated format such as 'QLD' will not match the full version of the name 'Queensland'. Before merging the two datasets, the values in stored in each key field need to be modified so that the format is consistent.

ACTIVITY 3.2.9



Open data is shared in a public format and can be downloaded and used by anyone for any purpose. A community application that has been developed using open data is the Qld School Zones app. Read the case study overview of the Qld School Zones app and answer the following questions:

- 1 Which open datasets does the Qld School Zones app use?
- 2 List the data requirements for this app.
- 3 Based on the data requirements you have identified, what type of data format would be best for the Qld School Zones app?
- 4 Locate and explore the open datasets used by this app.
 - a Identify any data anomalies.
 - b Identify data redundancy.
 - c Identify key fields used to link the datasets.
- 5 What type of data challenges do you think the developers may have encountered when developing this app?



3.2.16 Using education and transport to make roads safer for kids

Developing programmed components

The programmed components of a digital solution are responsible for controlling and processing inputs and outputs. Inputs can include user interactions, data collected from sensors, and event triggers such as timers or external systems. Outputs can include visual displays, stored data and actions of motors or other actuators. The programmed components are encoded with the rules of the system and control all the interactions between users and data stores in the digital solution.

Decomposing the problem

Decomposing a problem is a process of identifying the parts of the system that need to be developed. A visual diagram showing the movement of data through the system and interactions of the users will help to identify the logical modules of the system. Each module can then be further decomposed into inputs, processes and outputs until each process is simple enough to be represented as an algorithm.

A useful tool to decompose a digital problem involving users and data is a data flow diagram (DFD). A DFD is a graphical tool to represent the interactions in an information system. DFDs are explained in Unit 2. The context DFD is the overview of the system showing how the main components of the system interact. The central process from the context diagram can then be decomposed into a level 0 DFD, providing additional detail about the inputs, processes and outputs of the solution. Each process in the level 0 diagram can be further refined into level 1 diagrams. Further decomposition of each subprocess can occur until the system is completely decomposed. Usually only two or three levels of diagrams are required, but a complex system may require up to six or seven levels.

In order to identify which level DFD a process originates from, a simple numbering system is used, as shown in figure 3.2.18.

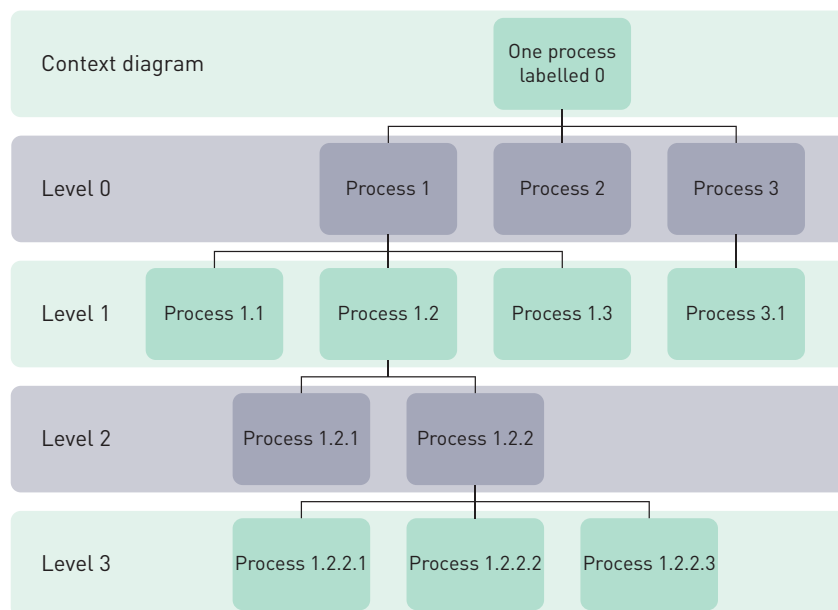


Figure 3.2.18 Numbering system for processes in a data flow diagram

Consider an automated checkout system. This system will interact with three entities: the customer, supermarket staff and technical support. The inputs to the system will include barcodes, weight sensors, payment cards and touchscreen interactions. Outputs will include messages and images displayed on the touchscreen, sounds, receipts, technical reports and activity reports. Data stores will include a database of product information, an activity log and an error log. The context diagram for this system is shown in figure 3.2.19.

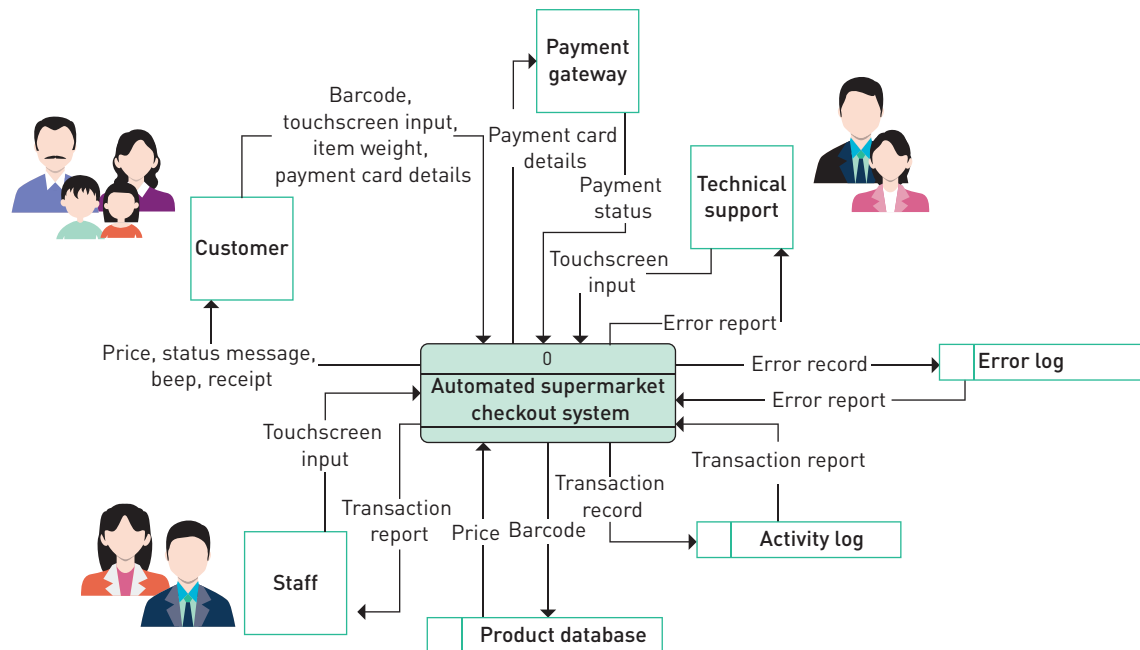


Figure 3.2.19 Context data flow diagram for an automated checkout system

At level 0, the individual functions of the system will be identified. They include 'scan item', 'make payment', 'override price', 'print transaction report' and 'print error report'. The level 0 DFD is shown in figure 3.2.20.

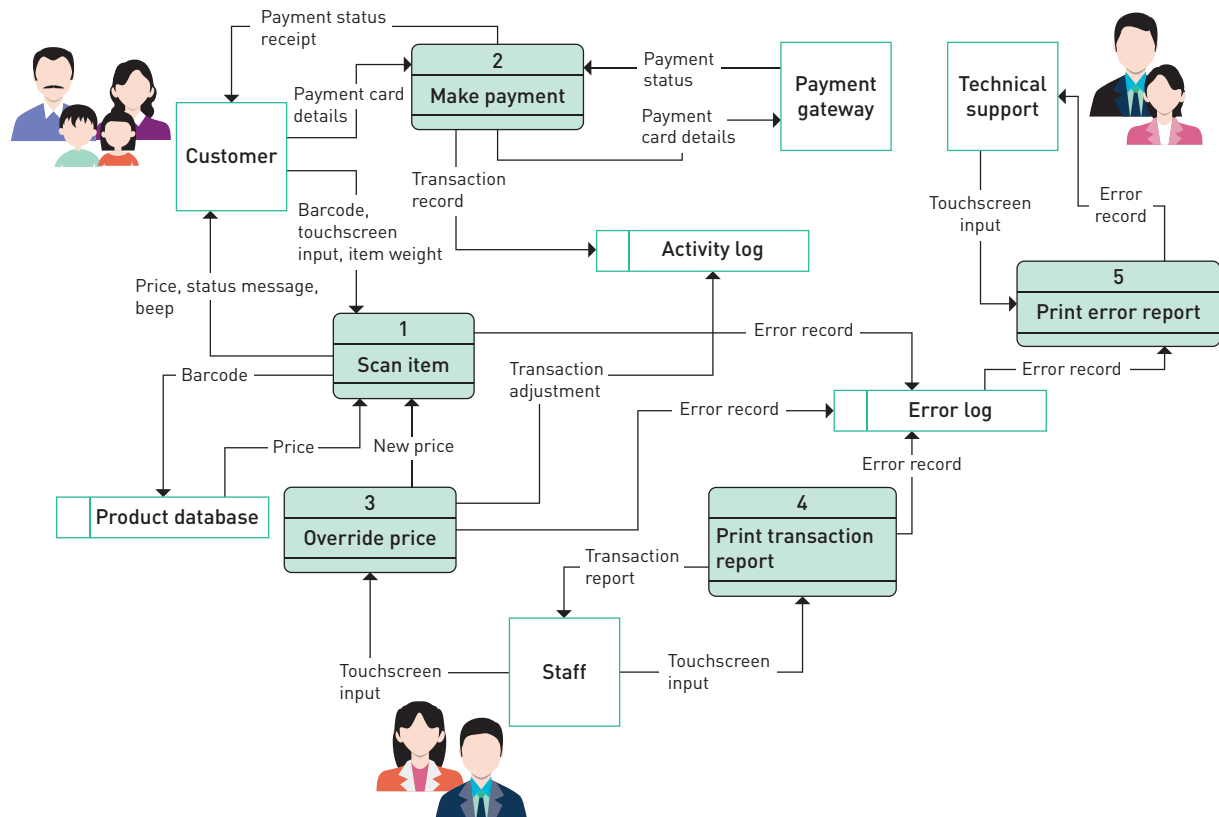


Figure 3.2.20 The level 0 data flow diagram decomposes the different functions of the system.

At level 1, the scan item process can be decomposed further to distinguish between general grocery items with a barcode and items, such as fruit and vegetables, that need to be weighed. The level 1 diagram is shown in figure 3.2.21.

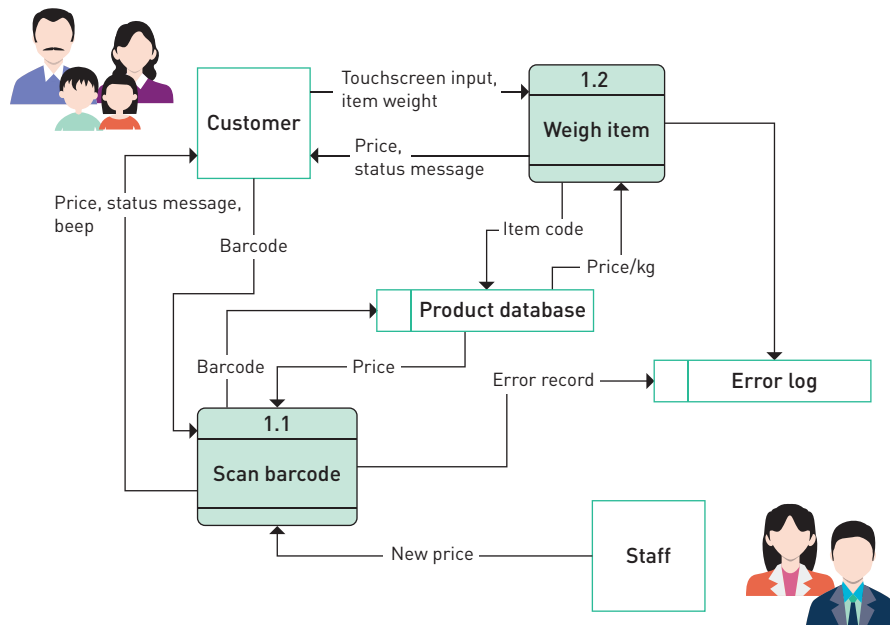


Figure 3.2.21 Level 1 data flow diagrams decompose a level 0 process.

When the system is fully decomposed, an algorithm can be written using pseudocode to detail the step-by-step instructions required for each process.

GROUP ACTIVITY 3.2.5

- 1 Form a group of three or four students and use an online collaborative tool such as Creately to complete the remaining DFD diagrams for the automated supermarket checkout machine.
- 2 Check each of the diagrams you have created to make sure they follow the DFD rules.
- 3 Compare one of your group diagrams with a similar diagram from another group. What criteria could be used to evaluate which diagram is more effective?
- 4 Consider the impact of adding an additional function to the system, such as a two-for-one offer. What parts of the system would change? How does a change on one diagram impact other diagram levels?



RULES WHEN CREATING DFDs

- 1 Use no more than seven processes in a single data flow diagram.
- 2 Each process must have at least one data flow in and one data flow out.
- 3 Every data flow must connect to at least one process.
- 4 Every entity must connect to at least one process.
- 5 Data flows in only one direction.
- 6 Data flow labels describe information, not actions.
- 7 Processes are numbered according to their level.
- 8 Process names are usually verbs.
- 9 Data flows and data stores are usually nouns.



Developing algorithms for programmed components

An algorithm is a set of instructions that can be translated into code. The DFD shows how the data flows between different parts of the system. The algorithm explains the logical steps that will process the data to transform the inputs into outputs. Individually, each algorithm is a plan for one part of the system. The main program module controls how these algorithms work together to create a digital solution.

Consider what happens when you start up a word processing program installed on your computer. When you double-click the desktop icon, the program is loaded into memory, the user interface is presented on the screen and then the system waits for the user to type or click something. When the word processing program detects a keypress or mouse click, it responds by loading the program to control the mouse click or keypress into memory. This program then evaluates the user interaction and loads the subprogram that relates to the area clicked or key pressed. Throughout this process the program is continually updating or accessing a number of different data stores, such as the active document, a list of changes made to the document (for use by the undo function), backup versions of the document (for the autosave function), dictionaries (for the spellchecker), thesaurus, image libraries and font libraries.

A simplified version of this process can be represented as an IPO (input, process, output) chart.

Input	Process	Output
X coordinate of mouse Y coordinate of mouse	Left-click Handler Algorithm: Identify area of click IF menu click THEN Activate drop down ELSE IF header click THEN Open Header panel ELSE IF text click THEN Move cursor to click area ELSE Do nothing ENDIF	Menu style Cursor x, y Viewing mode
Keycode	Key Handler Algorithm: Identify key pressed IF Ctrl_Key THEN Activate shortcut ELSE IF Function_Key THEN Activate function ELSE IF AlphaNumeric_Key Write key value at cursor position ENDIF	Char Command Code

In the supermarket checkout example, each process can be defined by an algorithm and an IPO chart.

Inputs	Process	Outputs
Barcode	Scan barcode Algorithm: <pre> READ Barcode OUTPUT Sound Beep SET sqlQuery TO SELECT Price FROM Products WHERE id=Barcode SEND sqlQuery TO Product Database RECEIVE price FROM Product Database IF price is 0 THEN SET error to TRUE SET ErrorRecord TO "Error: Barcode not found please scan again or ask for assistance" APPEND ErrorRecord to Error Log ASK for newPrice IF valid newPrice entered THEN SET error to FALSE SET price TO newPrice ENDIF ENDIF IF Error THEN DISPLAY Error Record ELSE DISPLAY "Price is"+ price ADD price TO total DISPLAY "Total is" + total ENDIF </pre>	Price Status message Beep

ACTIVITY 3.2.10

- 1 Create an IPO chart for the Weigh Item process.
- 2 Identify the main inputs and outputs from the DFD. These will usually be the inputs and outputs from/to an external entity.
- 3 Write an algorithm to show each step of the process.
- 4 Check that all the other inputs/outputs are represented correctly.



Refining the solution

Throughout the process of developing ideas for the digital solution there are constant refinements at each step. At any point in the process, you may discard the idea altogether or go back to an earlier sketch and redevelop from there. Once the idea is fully formed, you will have a collection of user interface sketches, paper prototypes, DFDs and IPO charts or algorithms. At this point you should refer back to the list of requirements you identified at the beginning of the process to check that the digital solution is fit for

the purpose intended. A simple checklist is a suitable way to evaluate if the digital solution meets the requirements. If any of the requirements are not checked off, then it is important to go back to the sketches and diagrams and add any missing functions or features.



3.2.17
Lorem
ipsum

Low-fidelity prototypes

The fidelity of a prototype is a measure of how exactly it matches the final solution. A sketch is a low-fidelity prototype of a graphical interface, paper cut-outs are examples of a low-fidelity prototype to test user interactions, and a wireframe template with buttons and 'lorem ipsum' text is a low-fidelity prototype for a web application.

Once you have established that the low-fidelity prototype of the solution meets the project objectives, further refinement is still necessary. How can the solution be improved? Is there a way to make it safer? More environmentally friendly? Less complex? More modular? More aesthetically pleasing? Target audience feedback and technical feasibility research will also play an important role in identifying improvements.

Low-fidelity prototypes are quick to produce so they can easily be thrown away, and are easy to adapt. Even when they are discarded, they have still provided the development team with valuable information about useability and utility of solution ideas. People are more likely to be honest about features they don't like with a paper prototype because they see that the solution is unfinished and easily influenced.

Lorem ipsum text is a convention used since the 1500s to show how given amounts of printed text will appear in a template or layout. Today it is used for online applications as well. These scrambled Latin paragraphs of text are unreadable to most modern users and help to focus attention on layout rather than content.



ACTIVITY 3.2.11

- 1 Review the Moneythink case study published by IDEO.org.
- 2 Explain how early feedback about ideas for the system influenced the design.



Explore



3.2.18 Case
studies

Pitching an idea

To obtain feedback about an idea for a digital prototype, it is first necessary to package the prototype in a format that can be quickly and easily understood by the target audience. A short video or presentation highlighting how the solution will be used, and showing mock-ups of the user interface, is a common way to test the market.

When developing a pitch:

- Try to summarise the essence of the solution in one minute.
- Share the general features of the solution without getting too bogged down in the details.
- Use an appropriate presentation tool to collect your ideas and present them professionally.
- Practise and refine your pitch until you can present it as a short version (1 minute), a medium version (3–5 minutes) or a longer version (30 minutes to an hour).

ACTIVITY 3.2.12

At Halfbrick Studios, a Brisbane game development studio, all members of the company are encouraged to pitch a new game idea several times a year on what is known as Halfbrick Fridays. Ideas for new games are selected from these pitches, and many become new game development projects. Some of the game ideas are never fully produced and are discarded after a few days or weeks. Others go through a full development cycle and become popular games downloaded by millions of users.

In this activity, inspired by the Friday pitch process, sketch a low-fidelity prototype of a new game idea and then pitch your idea to the class in a maximum of 3 minutes.

- 1 On your own, reflect on the recreational activities you have participated in over the past year and use your recreational experiences as inspiration to sketch ideas for a new game or mobile app.



Develop

- 2 Prepare a short presentation to explain your idea to the class.
 - a In the first minute, summarise your game idea and why it is needed.
 - b In the second minute, demonstrate your idea using sketches, a slideshow and/or a video.
 - c In the final minute, tell a story about someone who may interact with your game or app and the impact it could have on them.
- 3 Present your pitch to the class.
- 4 After the pitch, interview up to five classmates to obtain their feedback and suggestions for improvement.

TEST YOUR KNOWLEDGE

- 1 Describe an example of digital disruption.
- 2 Sketch the design thinking double diamond and annotate your sketch to explain how the terms 'divergence' and 'convergence' apply to the design thinking process.
- 3 Explain why it is important to defer judgement during the initial brainstorming phase of the design thinking process.
- 4 Define the term 'proto-persona' and describe one way proto-personas are used to develop ideas for an innovative digital solution.
- 5 Categorise the following requirements as functional requirements, non-functional requirements or constraints:
 - a Must be suitable for young children
 - b Must comply with accessibility requirements
 - c Should be compatible with Firefox and Safari
 - d Could be created using a colour scheme that matches the company logo
 - e Must use at least one open dataset
 - f Should check for user input errors
 - g Must protect personal information
 - h Could be created by a single programmer
 - i Won't access websites blocked by the organisation web filter software
 - j Should use open source software
 - k Must maintain data integrity
 - l Won't use copyright images
 - m Should appeal to Australians
 - n Could require an audit log
 - o Should follow the organisational style guide
 - p Must be deployed in 4 weeks
 - q Could include user login screen
 - r Won't have spelling errors
- 6 Describe the advantages and disadvantages of using online surveys to analyse target user preferences.
- 7 Sketch a simple data flow diagram that includes at least one external entity, process, data store and data flow. Annotate the sketch to show how the DFD rules have been applied.
- 8 Describe one type of diagram used for representing the programmed components of a digital solution.
- 9 What are the five useability principles?
- 10 Explain the term 'useability' using at least three examples.
- 11 Describe three advantages of low-fidelity prototypes.
- 12 Describe 10 of the elements and principles of design.
- 13 Explain why good aesthetics is an important useability consideration.
- 14 Describe two common data formats used in digital solutions.
- 15 Explain the benefits of using a relational database to store and access data for an innovative digital solution.
- 16 Explain what is meant by 'decomposing' a problem.
- 17 Describe a method for symbolising the components of a digital solution.
- 18 Explain what is meant by 'cleaning' data.
- 19 Describe three important steps to follow when merging data sources.
- 20 Describe the purpose of pitching an idea for a new digital solution.



APPLY YOUR KNOWLEDGE

The following project ideas provide students with an opportunity to apply the concepts from this section to a specific technology context.

TECHNOLOGY CONTEXT	PROJECT IDEA
Mobile applications or web applications	Do you want an app with that?
Interactive media or intelligent systems	Everyone wins a prize!

Do you want an app with that?



Figure 3.2.22 Many restaurants today use technology to manage the ordering process.

As shown in figure 3.2.22, many restaurants today use technology to manage the ordering process. For example, a group of friends walk into a restaurant and are directed to one of the empty tables. After a while, a waiter arrives with a tablet device in his hand, ready to take the table order. He logs in to the restaurant system on an iPad and selects the table he is taking an order for. As the customers place their order, the waiter enters the details by tapping options on the tablet. Any menu variations or food allergies are recorded on the order by making the appropriate selections on the iPad. As soon as the order is complete, the waiter wirelessly sends the order to the kitchen by tapping the send button on the table waiter app and the order is instantly prioritised and displayed on the screen of the appropriate chef's workstation. When the order is ready to be collected, the chef taps an 'order ready' button on the kitchen display and the waiter comes to collect the order and delivers it to the customer. When the group of friends are ready to leave, they head to the cashier to pay for their order. Somehow,

while they were eating, their order details and bill were sent to the point-of-sale system.

- 1 Design the user experience for the table waiter app. Create a paper prototype for the waiter and kitchen user interfaces.
- 2 Develop the data stores for the restaurant. List the data that will need to be stored for this system. What type of file format would be suitable for each data store? What data items will be stored in each data store? Will any of the data stores need to be merged?
- 3 Sketch a context DFD diagram for the table waiter app. Include the customers, waiter and kitchen as external entities and the data stores you identified in step 2. Add the data flows and decompose the context diagram in to level 0 and level 1 diagrams. Check the DFD rules have been followed in each diagram.
- 4 Write an algorithm for each of the processes in the DFDs to show what calculations need to occur to change the input data into a suitable output format.
- 5 Complete a walk-through of the system using the paper prototype, data definitions and DFDs. Simulate the order process, recording data on sticky notes and moving the data through the paper prototype system. Desk check the algorithms to record how inputs are processed and formatted as outputs.
- 6 When the walk-through is complete, interview the people who played the roles of the different types of users during the simulation. Record and summarise their feedback.
- 7 List errors you identified in the walk-through and make recommendations for how to fix them. In particular, check the following:
 - a Did each user interface include areas where the required data could be collected?
 - b Was the user able to find and use the buttons and data fields easily?
 - c Did each data store include a field to store the data from each data flow?
 - d Was any data stored but not used?
 - e Was any data needed that was not stored?
 - f Did the algorithms process the data as expected?
 - g Which algorithms were missing?

Everyone wins a prize!



Figure 3.2.23 Many mobile app and virtual reality games simulate the sideshow experience.



Figure 3.2.24 This IoT project showcased on hackster.io uses Particle Photon and Adafruit trinket microcontrollers to build a homemade laser tag game that dispenses lollies.

Every year at the Brisbane Ekka and many similar regional shows, fetes and festivals, sideshow alley games attract millions of people from a wide range of backgrounds and ages. Use design thinking to brainstorm ideas for a real or virtual innovative new sideshow alley game that incorporates data about the player into the gameplay. You may be inspired by the projects illustrated in figures 3.2.23 and 3.2.24

as a starting point, but remember to use design thinking to brainstorm a wide range of crazy ideas.

- 1 Use divergent thinking to collect a wide range of possible and impossible ideas for digital solutions related to a sideshow alley.
- 2 Apply convergent thinking to narrow the options to something that fits within your chosen technology context and can be implemented with materials available to you at school.
- 3 Sketch a rough diagram identifying possible user interface, data and programmed components of the solution.
- 4 Define the user requirements and conduct online surveys or user interviews to test and refine early ideas.
- 5 Define the data requirements for the player information and consider how the data about the player will be collected and stored. Identify the most suitable data format for player data.
- 6 Collect sample data and explore how it is represented in the selected technology context. Refine data requirements based on sample data collections.
- 7 Create a DFD showing how the data will flow through the system and identifying the programmed components of the solution.
- 8 Create an IPO chart to describe the high-level algorithms for each process.
- 9 Desk check the algorithms to record how inputs are processed and formatted as outputs.
- 10 Create a pitch to present your idea and obtain feedback from the target audience. Record and summarise their feedback.
- 11 Evaluate improvements suggested during your pitch and make recommendations for how to incorporate them in your design.
- 12 Evaluate your proposed digital solution to check that it meets the original requirements. In particular, check the following:
 - a Will the game appeal to a wide range of people from different backgrounds and ages?
 - b Is the solution innovative?
 - c Does the solution incorporate data about the player?
 - d Can it be implemented with materials available at the school?
 - e How can aesthetics or design principles be used to improve the user experience?
 - f Should the player data stores be protected?

3.3

GENERATING A DIGITAL PROTOTYPE



Generate

In this section you will use computational thinking to synthesise information and ideas and create programmed components, user interfaces and databases. You will establish a development environment that allows you to demonstrate how the different components of your prototype solution interconnect to solve a problem. You will run simulations and collect data about the accuracy and efficiency of solution components.

Project management approaches

A project involves all four phases of the problem-solving process. The need for the project is identified during the explore phase and the specific requirements of the project are determined during the develop phase. During the generate stage, the focus shifts from many possible directions to one preferred outcome. When there is a clear objective and defined requirements, a project management approach is helpful to keep the project team on track. Evaluation processes are a key component of many project management approaches.

There are many different approaches to project management, and these are often combined in hybrid models that combine the best parts from different approaches to suit the characteristics of a particular project.

Waterfall model

The 'waterfall' project model is best for linear projects that have a fixed set of requirements that are not likely to change. It was first proposed in 1970 at an engineering conference with the aim of providing advice about how to deliver a large software project on time and within a budget. It is still popular for large projects with a fixed set of functional requirements that need to be delivered by a specific date for a specific cost. It is also popular for software engineering projects where human safety is a high-priority requirement, such as control systems for aircrafts, elevators and space travel.

The waterfall model outlined in figure 3.3.1 has the advantage of being easily understood, easy to manage from a developer perspective and with a predictable outcome. A lot of time is spent at the design stage to ensure that all aspects of the implementation phase can be measured and costed accurately. Every requirement is defined in enough detail to predict the implementation time and costs. Any change to the requirements filters through the subsequent linear stages and needs to be analysed, costed, documented and agreed as an addendum to the original contract, incurring additional time and cost penalties for the client. The main disadvantage of the waterfall model is that it takes a long time. The rate of change of technology often means that, in the time that a complex system is being developed, the original requirements and underlying technology changes significantly. In short, the model does not easily adapt to changes in user requirements, business requirements, economic climate or technological advancement. For this reason, many businesses now favour agile project management approaches, particularly in mobile and web application development projects, over the waterfall model.

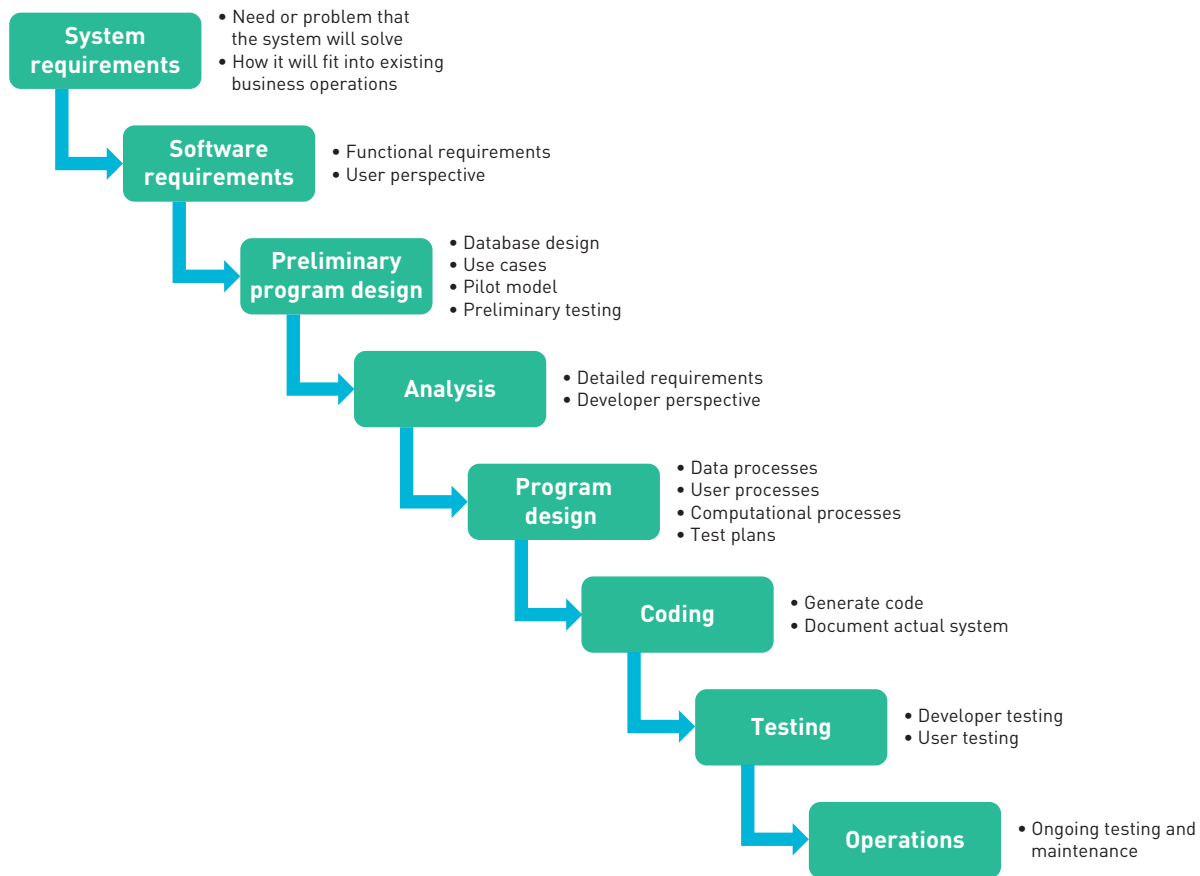


Figure 3.3.1 The waterfall model proposed by Royce in 1970 is still relevant today.

Source: Based on diagram by Winston W Royce, 'Managing the Development of Large Software Systems', 1970, Fig. 2, in *Technical Papers of Western Electronic Show and Convention (WesCon)*, August 25–28, 1970, Los Angeles, USA.

Agile project management

In 2001 the 'Agile Manifesto' was published by a group of 12 software methodology leaders who were frustrated with traditional approaches to software development. They named themselves the Agile Alliance and agreed on the four core principles shown here (from the Agile Manifesto).

Since then, a number of agile approaches, including Scrum and Extreme Programming, have increased steadily in popularity. Many traditional organisations now use agile management approaches for some types of projects that were previously managed exclusively by a waterfall model. IBM uses an agile approach called the Rational Unified Process, which values Unified Modelling Language (UML) diagrams rather than wordy explanations as a way of documenting each component of the solution.

Agile project management approaches have the following elements in common:

- They are responsive to changes in user requirements and can adapt quickly to incorporate new technology.
- They promote focused incremental change based on regular review and evaluation.
- They evolve over time with unpredictable pathways and end dates and therefore are considered risky compared with traditional approaches.
- They require teamwork, collaboration and frequent communication between users and developers.

AGILE PRINCIPLES

- **Individuals and interactions** over *processes and tools*
- **Working software** over *comprehensive documentation*
- **Customer collaboration** over *contract negotiation*
- **Responding to change** over *following a plan*



3.3.1 Zoho sprints

A number of online collaborative tools have been developed to support agile project management; according to PCMag, one of the most popular is Zoho Sprints. Many of the agile processes illustrated in this section are screenshots from the Zoho Sprints website.

Before using an agile project strategy, it is important to understand some agile terminology.

User stories

In agile terminology, 'user stories' are the list of requirements from the user's perspective. Stories such as those listed in figure 3.3.2 are prioritised, with the most important stories at the top of the list. The idea of user stories comes from the Extreme Programming and Scrum methodologies. Unlike in the waterfall model, the requirements do not have to be decided up front. A few simple key ideas are represented in the format 'As a [user role], I need to [achieve a goal] so that I can [reason for activity]'. These user stories then become conversations between the user and the developer, during which detailed notes are written on the back of the story card to remove ambiguity. As the project evolves, user stories are reviewed regularly and, if necessary, can be removed or modified, or more user stories can be added. To prioritise user stories the MoSCoW method can be used, replacing the phrase 'I need to' with 'I must', 'I should', 'I could' or 'I want to'. It is important to keep user stories simple and related to a single action. Epic stories need to be decomposed into smaller, bite-sized stories that relate to a single function or feature.

Sprint

The concept of a 'sprint' comes from the Scrum methodology. Its aim is to deliver incremental stages of the project to the client at regular intervals and obtain ongoing client and user feedback. A sprint is a mini-project that produces a working iteration of the software. It differs from the prototyping approach used in engineering in that, while a prototype is usually thrown away, the output from a sprint is not. The first sprint becomes the kernel of the final system, and future iterations build additional layers on top of this core. The duration of each sprint is governed by the time box, which is usually 2–4 weeks, and its progress is tracked and reviewed daily on a team board. The sprint is developed by a small team of developers called the 'scrum', who are self-managing but are supported by a 'scrum master', whose main purpose is to remove any distractions or roadblocks that impede progress towards the end goal. The need for good communication between users and developers during the sprint is so important that the user or client representative may be encouraged to sit with, or become part of, the project team. A typical project usually comprises between six and 10 sprints in total.

#	User story title	User story description	Priority	Notes
1	Facebook Integration  JIRA MDT-13	A user wants to sign up via Facebook	MUST HAVE	- We will need to talk to Cassie Owens about this one. - There has also been some research done on this (see Facebook integration prototype)
2	Activity Stream  JIRA MDT-14	A user wants to view the latest updates via the mobile dashboard so that they can get a better understanding of what is in place	MUST HAVE	
3	Post Updates  JIRA MDT-15	A user wants to be able to post status updates on the go	MUST HAVE	The key things we will need to support: - Text status updates - Mentions - Support for images - Smart embedding for things like YouTube videos etc.
4	API  JIRA MDT-16	A developer wants to integrate with the mobile app so that they can embed the activity stream on their website	SHOULD HAVE	We should chat to Team Dyno as they did something similar.

Figure 3.3.2 User stories describe requirements from the user's point of view.

Source: <https://www.atlassian.com/agile/requirements>

Backlog

The ‘backlog’ is the list of tasks that have not yet been started. In most situations it will be the list of user stories. After the user stories have been collected and discussed, the developers build a prioritised list of tasks needed to create the functionality required. Each task is given an estimation value (based on the time developers think it will take to implement the feature) and a priority (based on the priority of the user story that relates to the task). When all tasks have been estimated and prioritised, work begins on the first sprint. At the beginning of a sprint, tasks are moved from the backlog to the sprint team board. During the sprint, some tasks are completed and closed, and new tasks may be identified and added to the team board. Any unfinished tasks on the team board at the end of the sprint are returned to the backlog.

Burn down and burn up

These terms are associated with the charts used in the Scrum methodology to show progress towards completion of tasks. The number of tasks at any particular point is fluid, but it is still the best way to measure progress towards the project end point. ‘Burn down’ charts are used to identify the percentage of unfinished tasks and predict a possible end-of-project date, based on the estimation points for each task and the task velocity (the rate at which tasks are completed). The ‘burn up’ is the number of tasks that have been completed. It is used to compare task estimation points with actual hours worked, and thus to measure team productivity.

GROUP ACTIVITY 3.3.1

Use the Zoho Sprints online tool to create a new project for the table waiter app you began in the previous section.

Invite two classmates to be part of your project team. Each member of the team should consider the role of one of the user types for this app. Select Create/User stories, and build a list of user stories that relate to the waiter, chef and restaurant owner user roles.

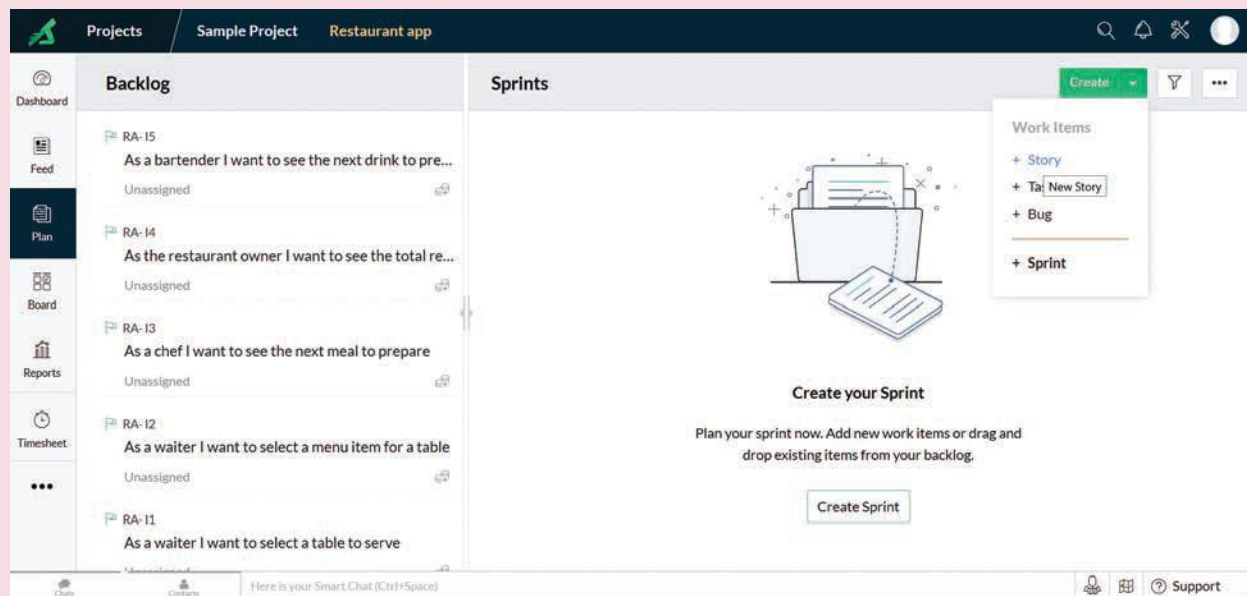


Figure 3.3.3 Creating user stories in Zoho Sprints

Each user story is automatically added to the backlog.

Create a new sprint for your team and have each team member drag one of the user stories to the sprint.

Select your user story by double-clicking it. Edit the details to allocate a suitable priority and estimation points. Create subtasks related to the user and data interface for the story.



3.3.2 Zoho
sprints



Generate

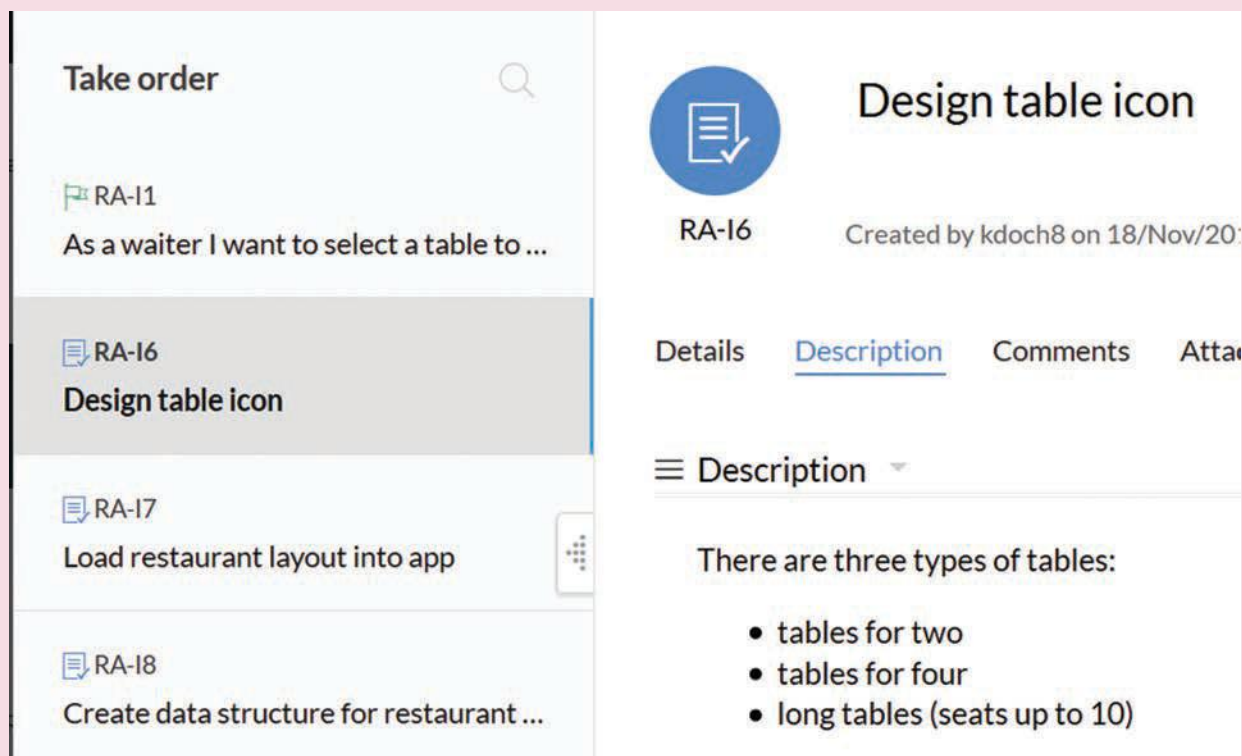


Figure 3.3.4 Describing and prioritising subtasks associated with user stories in Zoho Sprints

Establishing the development environment



3.3.3
Robotics and
autonomous
systems
group

In Unit 1, you learned about the importance of using an integrated development environment (IDE) to develop program code. In Unit 2, you learned about different types of database management systems that can be used to manage a database. In this unit you will need to identify a collection of programming development tools that allow you to design user interfaces, create program code and generate data components. Your teacher may have some tools in mind that are suitable for use at school, but there are likely to be some constraints within your school or home environment that limit your development options. Your choice of technology context will also determine the types of tools that are most suitable for your project. Software development organisations often use the same set of tools for every project, but it is useful to explore different options as new types of tools may help you to work more efficiently.

Using criteria to evaluate programming development environments

When selecting a programming development environment or tool it is important to consider the following criteria:

- Does it contain the minimum features required?
- Is it easy to learn?
- Does it include helpful debugging features?
- Has it been used successfully before for similar projects?
- Are there tutorials, discussion forums or sample code to help me if I get stuck?
- Is it free for student use or are there hidden costs?
- Will it run on the equipment I use at home and at school?

Table 3.3.1 lists some development tools that may be suitable for projects you generate during this unit.

Table 3.3.1 There are a range of integrated development environments suitable for generating digital solutions.

WEB APPLICATIONS	MOBILE APPLICATIONS	INTERACTIVE MEDIA	INTELLIGENT SYSTEMS
XAMPP	Microsoft Visual Studio	Microsoft Visual Studio	Microsoft Visual Studio
Netbeans	OutSystems	Light Table	Atom/PlatformIO
Microsoft Visual Studio	AIDE	Thonny	IBM Watson
Light Table	Xcode	Unity	Losant
Brackets	IFTTT	Adobe Animate	Dronesmith
Atom	Cordova	Processing	IFTTT
PythonAnywhere	Xamarin	Python with Pygame or Arcade library	ROS
Aptana			

Your development environment for Unit 3 may consist of a single development tool or a collection of tools. The combination of tools you select will need to support, as a minimum:

- a procedural programming language
- connections to a database including SQL
- reading and writing flat files or reading and writing streams of data from/to IoT components
- creation of graphical user interfaces.

ACTIVITY 3.3.1

Explore the possible programming development environments for one of the technology contexts for Digital Solutions.

- 1 Use the list of criteria provided above as a checklist to evaluate some of the environments that you think may be suitable for your next project.
- 2 Choose the environment that best meets your requirements. Establish access to the environment on your home computer and/or school laptop. You may need to download and install software, or set up and configure an account profile online.
- 3 Complete the first column of the table below to brainstorm what you already know about this tool or environment.

WHAT I ALREADY KNOW	WHAT I WAS ABLE TO FIND OUT	WHAT ELSE I WOULD LIKE TO KNOW

- 4 Explore the programming environment by creating a test project, accessing help, reading a tutorial or interacting with other users on a discussion forum. Complete the second column in the table above to record what you learned.
- 5 Reflect on the types of activities you will be using the programming environment for, and complete the third column to brainstorm a list of features you may need time or help to investigate. Talk to your peers and teachers about the best way to find out about these features.



Generating an interactive data-driven digital prototype

Generating a system that integrates user inputs and data stores with coded processes requires a great deal of computational thinking:

- 1 Decompose the problem.
- 2 Look for patterns and possible solutions.
- 3 Build an abstract model of the solution.
- 4 Create algorithms to show how individual components will operate.

You have already decomposed the problem when you determined the requirements for the system. You identified possible solutions as you developed ideas and information about the user interface and data elements. You created an abstract model when you developed data flow diagrams to represent interactions between user and data components. This section will focus on generating algorithms to show how individual components will operate.

If you are working in a team, use a project management strategy to divide the work among members of the team. Allocate a different module to each team member and build a simple working prototype with minimal functionality. The first iteration of the prototype may simply display a message stating what each module will do. Gradually add to the functionality of each module throughout the project. Making small, incremental changes, frequently tested for interoperability with other modules, is a more successful approach than trying to build the entire system at once.

A programming project is a never-ending process of exploring and developing ideas, generating user interfaces, data and code, and evaluating the user and developer experiences to recommend improvements. Perseverance and collaboration are important ingredients for success. Debugging is much easier if you start with a simple working prototype and build it up gradually, adding small changes to each module until the solution begins to take shape.

Generating readable modular programs

When programming as part of a team, it is possible that you will generate code from an algorithm that someone else has written. It is also possible that a program or function that you generate will be used by other members of the programming team. For this reason, it is important to ensure that algorithms are represented using common pseudocode conventions, and that they are detailed enough or unambiguous enough to ensure that they can be interpreted the same way by other members of the programming team. When the program is created, it is important that the comments within the program code show how the algorithm has been interpreted so that other programmers can easily correct mistakes or make changes to the code.

Pseudocode conventions

The common conventions for writing pseudocode are outlined in the Digital Solutions syllabus and are explained on page 41 in Unit 1.

Many beginner programmers make the mistake of representing algorithms using code-specific algorithm steps. Following algorithmic conventions helps to focus on the logic of the process, rather than on the way it will be coded. The programmer can then decide the best way to represent the algorithm as code.

Table 3.3.2 Good and bad examples of algorithms	
BAD EXAMPLE	GOOD EXAMPLE
do print(i) while (i++<10)	set counter to 0 REPEAT add 1 to counter print counter UNTIL counter = 10

When generating a program module from an algorithm, it is good programming practice to use comment syntax to represent each algorithm step, as shown in figure 3.3.5. This makes it easier for other programmers to follow the program logic.


```

public boolean moveRobot (Robot aRobot) {
// IF robot has no obstacle in front THEN
    // Call Move robot
    // Add the move command to the command history
    //RETURN true
// ELSE
    // Return false without moving the robot
// END IF
}

```

Figure 3.3.5 It is good programming practice to include the algorithm as comments in the source code.

Removing ambiguity

When writing a programming component that interacts with other coded components, it is helpful to include a detailed comment at the beginning of the function or program header explaining the purpose of the program, the format required for each input and the format of each output. This will help to ensure that the data passed between the components is interpreted correctly. Figure 3.3.6 shows the function header comment for a program that processes a keyboard buffer.

```

/*F*****
int KB_Getline ( pKbdBuf, MaxChars )

char *pKbdBuf;    /* - > buffer for keyboard chars */
int MaxChars;     /* max chars to read before beeping */

/*
* PURPOSE      :      Input line of text from keyboard
*
* RETURN       :      VALID_DATA = valid read
*                  (see keyboard.h for the rest)
* NOTES        :      unknown characters returned as ``
*      Backspace is the only editing allowed
*F*/

```

Figure 3.3.6 Detailed code comments at the beginning of a program module or function remove ambiguity.

When developing the program components for a digital solution, it is important to specify the format of the data that is passed between processes or functions. An appropriate time to do this is when writing the algorithm for program components. Consider the example of processing user input.

Figure 3.3.7 shows that a function named `get_input()` will collect user input and pass the data entered to the `process_input()` function. The `process_input()` function receives data from `get_input()` and processes it before storing it as user profile data. The purpose of this concept

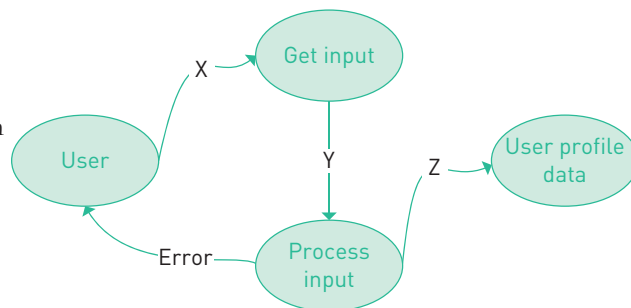


Figure 3.3.7 This concept map identifies that `get_input()` and `process_input()` functions will be required in the digital solution.

map is to analyse possible program components, but the data type of the value `y` is ambiguous at this stage. For example, the user may enter data using a keyboard, a barcode scanner or a touch to the screen. If the `process_input()` function needs to perform arithmetic calculations with the user input as shown in the IPO chart below, the value will need to be converted to a numeric data type.

Table 3.3.3 Ambiguous IPO chart

Input	Process	Output
y	<pre> IF y > 14 THEN SET z TO y OUTPUT z ELSE SET error TO "Input error" OUTPUT error ENDIF </pre>	<pre> z error </pre>

The IPO chart shown in table 3.3.3 does not consider the data type of the input value y. The developer must decide whether it will be the responsibility of the `get_input()` function or the `process_input()` function to manage this conversion and remove the ambiguity, either in the algorithm (as shown in table 3.3.4) and/or as comments in the program code. Specifying the purpose and data type of inputs and outputs will remove ambiguity and improve interoperability of program modules.

Table 3.3.4 This IPO chart specifies data types to remove ambiguity.

Input	Process	Output
y (string)	<pre> SET z TO int(y) IF z > 14 THEN OUTPUT z ELSE SET error TO "You must be over 14 to create a user profile." OUTPUT error ENDIF </pre>	<pre> z (int) error (string) </pre>

The data passed between program functions is referred to as function parameters. Strongly typed programming languages require a specific data type to be specified for each function parameter as part of the programming syntax. Other programming languages do not require the data type to be specified, allowing more flexibility and more ambiguity in the way the program manages that data. When programming a function, if the programming language does not force the programmer to specify the datatype of function parameters, it is important to include code comments in the function header indicating the range of valid data types the function can handle. Appropriate error messages or error codes should be returned if data of an incorrect type is passed to the function.

Pair programming

It is important for functions and program modules that will be shared by other parts of the program to be accurate and efficient. An error or inefficiency in a common program module will impact every other module that interacts with it. When prioritising debugging activities, testing and refining reusable modules should be top of the list. One approach to coding and debugging important program modules is called pair programming. An example of pair programming is shown in figure 3.3.8. Typically, two programmers work side by side to develop a single module. One programmer types and the other checks that the line of code is free of syntax or logic errors. From time to time, the programmers swap roles. Pair programming is part of the agile project methodology and has been shown to create high-quality code in a short time frame.

The pair programming technique can also be applied to writing algorithms. Early computer programmers regularly asked other programmers to check their algorithms before they were coded to reduce the possibility of expensive mistakes.

Coding for user interactions

Digital solutions present information to users and respond to user input. The method for user interaction will depend largely on the technology context and the hardware input device. In web applications, the user display will likely be a responsive web page developed with HTML and styled with CSS. User input will be managed with button clicks, keyboard entry and mouse clicks. In mobile applications, user information can be communicated through a small mobile screen, sounds, vibrations and alerts. Input is likely to occur primarily through the touchscreen or, in some cases, via the microphone or internal phone sensors. Intelligent systems will provide a wide range of possible user interactions such as motion sensors, buttons, heat, moisture and sound detectors, but graphical display options may be limited unless they are linked to a web page. The interactive media technology context provides a wide range of information display options from high-definition monitors to virtual reality experiences. Traditionally, user input for interactive media has been provided through keyboard and mouse devices, but game controllers and gesture controls provide alternative methods of interaction. When coding for user interaction, the main considerations are to communicate effectively with the user to encourage an appropriate response and to verify the user input data to check the response provided falls within the expected range.



Getty Images Plus/E+/Tomml

Figure 3.3.8 Two heads are usually better than one.

Communicating effectively with the user

The developer needs to generate code to display information and messages to the user for every possible hardware device the user may use to access the digital solution. Due to the wide range of possible interactions, it is helpful to identify a single popular hardware device as a starting point to manage the scope of the project, and gradually add adaptations for other devices.

The interaction between the user and the computer will usually involve the sequence shown in figure 3.3.9.

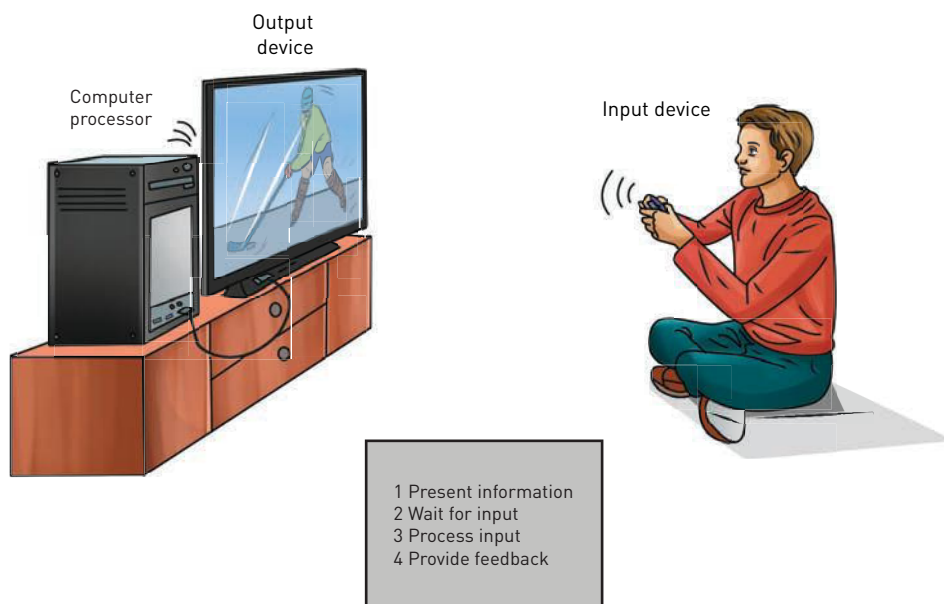


Figure 3.3.9 Sequence diagram showing the interaction between the user and the computer



ACTIVITY 3.3.2

Sketch a user interface for a login page for the sideshow alley carnival game you began in section 3.2.

- Sketch the welcome display for the game, including login instructions to the user.
- Sketch the display that will be shown if the user logs in correctly.
- Sketch the display that will be shown if the user doesn't follow the login instructions correctly.

Presenting information to the user

In most situations, information is presented to the user in the form of a graphical display. There are other ways of communicating information, such as through sound or vibrations, but the majority of user interactions in most technology contexts will be via a screen or other type of display. The basic layout of the display could be created with a generic template or user interface design tool. Dynamic components specific to the digital solution will need to be coded and will change in response to input. The layout for the interaction display

should be consistent and uncluttered. Menus, buttons and input fields required for user input need to be clearly identifiable and the developer should consider which parts of the layout will be used to communicate information or error feedback to the user. As many actions as possible should be simplified and automated.

Waiting for user input

Waiting is a paradoxical concept for a computer. Computers never wait. They constantly process millions of instructions per second, regardless of whether there is something for them to do or not. In between the time when the digital solution presents information to the user and when it receives a response, it checks or listens to every sensor constantly to see if a response has been received. There are a number of ways to program a computer to wait:

- **Input sequence**
The next command in the program will not run until the user input is received. The computer checks constantly, millions of times per second, until the input is received.
- **Input loops (iteration)**
The computer asks for input and sets a timer. When the timer goes off, it checks if the input has been received. If the input is not received within the set time period, it asks for it again in a never-ending loop until the input is received or the user quits the program.
- **Event handlers**
A function called an event handler is attached to an event in such a way that the code will only run when the event occurs. An event can include a keypress, a mouse click, a timer tick, a change to a value of a field on a web form, or any other detectable event. This is called adding an event listener, or event-driven programming.

Processing user input

When the user responds with input, it is initially stored as a variable value in the computer memory. The program accesses the data and processes it using the programmed actions. It is important not to assume that the user has provided a valid input. Errors should be checked for, and handled gracefully. All user inputs must be converted to a digital format before they can be stored. Make sure the field sizes for any text entry are the correct size for the largest possible valid user response. It is a good idea to write code to echo the user data to a console display during testing to verify assumptions about the digital format used.

Providing feedback to the user

Error messages, sounds and colour alerts are important ways to provide feedback to a user about their response. When designing the user interface, it is important to plan how this feedback will be communicated to the user. Strategies for communicating errors to users include pop-up alerts or a display area shown on the screen. If you will be using the screen to display errors, it will be important to leave room in the display for the errors.

Validating user data

Computers are predictable but humans are not. **Data validation** is important to reduce the possibility of human error during user interactions with a digital solution. **Data verification** attempts to ensure that stored data that could be used in subsequent calculations is not only correctly formatted, but also accurate. Undetected data input errors multiply when incorrect data is distributed to other parts of the system, or to external systems. It is important to incorporate appropriate data validation methods when coding for user interactions.

Preventing user input errors

The design of the user interface can be done in a way that prevents some user errors. Some considerations for preventing errors are explained below.

- **Clear instructions**
Describe the expected format of the data. If you are expecting the user to enter their date of birth, make it clear whether this is in a dd/mm/yy or mm/dd/yyyy format. If you need the user to enter a delivery time, what time zone does the time relate to? Use non-technical language that the user can understand when providing instructions, and test that these instructions were interpreted correctly during user testing.
- **Input constraints**
Provide limited options so users must select a valid option. This could be in the form of a drop-down box, radio buttons or calendar widget, or by displaying a number of options the user can select using touch.

HTML5 introduced a number of input types to assist with validating user input, such as *range*, *email*, *date*, *tel* and *week*. HTML5 also introduced the *required* property for input tags to help manage empty form fields using browser software. Accessibility features such as *aria-required="true"* communicate input errors to assistive technologies. For more information about accessibility considerations for validating data through web interfaces, you can refer to online web tutorials.



3.3.4
W3C web
accessibility
tutorials

- **Proofreading**
Confirmation screens encourage users to check and confirm their inputs before sending them through to the application. Confirmation screens are also a great way to give the user an opportunity to undo/redo, and improve a user's feeling of control.
- **Double entry**
Sometimes it is important to provide the user with an opportunity to enter information twice and then compare both entries to check for inconsistencies. This is helpful for data that must be correct, but is difficult to verify in other ways, such as a password.

Validating user input

Providing immediate feedback to the user about their data entries gives them an opportunity to correct data that does not meet expected formats. It is important not to make assumptions about the data users have entered, and to test for both expected and unexpected inputs as well as missing inputs, data types and data patterns. Some important data validation checks to code into the user interface are explained below.

- **Missing input**
Test if the field is required but empty. For example, using the HTML5 *required* property in a web form or `[fieldname] != ""` in a desktop application.
- **Length**
Test if text input is the expected length. For example, a membership ID may need to contain an exact number of characters. All programming languages have a simple function or property that can be used to check the length of a data string.
- **Range**
Test if numeric input is in the expected range. For example, a high school student will have an age between 11 and 18, and a month will be between 1 and 12.



- **Allowed list**
Check if the value entered appears in a list of expected values, such as days of the week or states of a country.
- **Data patterns**
Use a regular expression to test if the input matches a specific pattern. For example, the regular expression `[A-Z]{3}[0-9]{3}` matches a registration plate in the format AAA999. Most programming languages have a regular expressions function or library to allow the use of regular expressions for pattern matching. RegexOne provides a simple online interactive tutorial to learn how to use regular expression syntax.
- **Check digit**
Barcodes, ISBNs and credit card numbers include a check digit. This can be used to ensure that a product barcode, library book or credit card has been scanned successfully. A check digit is calculated for a long number by an algorithm and appended to the end of the number. To ensure that the number entered by the user is valid, the leading digits are used to recalculate the check digit, which is then compared to the last digit entered by the user. If the calculated check digit matches the check digit entered by the user, the number has a high probability of being correct. Some examples of check digit calculations are explained on the CodeProject website.

Verifying user input

Even after completing all these data validation checks within the user interface code, it will still be possible for users to enter data that is not correct. An email address might be in the correct format but could still be incorrect. A student ID might be in a correct format, but the student may not yet exist in the system. For web applications, older browsers may not run the validation checks correctly allowing the user to submit invalid data to the server. It is often necessary to repeat validation checks within the data module of a digital solution. In particular, contact data that will be used for future communication with the user, or key data used to look up related information, should be verified.

- **Confirmation codes**
Some applications will send a code to the user using email or SMS to verify a mobile phone or email address data entered by the user. The code may be generated by a random number generator and sent to the user for manual entry, or embedded within a weblink that processes the confirmation when clicked.
- **Lookups**
It may be necessary to use an SQL query to verify that a value entered by the user exists in a database. Sometimes this method is used to SELECT data from a table to create customised drop-down lists based on a key data value entered by the user.

ACTIVITY 3.3.3



- 1 Sketch a user entry form that could be used to collect user profile data for the slideshow alley carnival game or another project of your choice.
- 2 Annotate the form to show the error prevention, data validation and data verification techniques applicable to each field.
- 3 Compare your design with another class member's design and work together to create a better design that combines features from the two sketches.
- 4 Reflect on the data validation techniques discussed in this section and answer these questions.
 - a How can user data be used to personalise a user experience?
 - b What is the impact of storing incorrect data?
 - c Who should be able to view or modify data entered by a user?

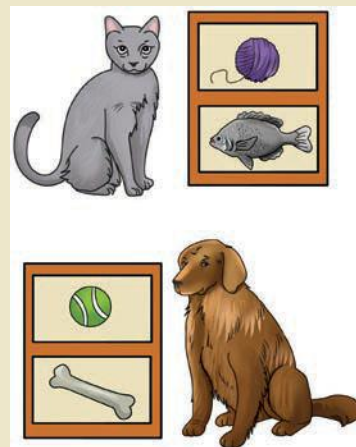


Figure 3.3.10 User profile data can be used to personalise the user experience.

Coding for database interactions

In this unit, the digital solutions you generate will require code that reads and/or writes to tables in a database. Regardless of your technology context, the database will need to be managed by a local or remote database management system (DBMS). The database platform you selected in the explore phase of your project will determine the specific libraries or functions required to interact with the DBMS and the syntax required will be defined in the API for your specific DBMS. The general process to read or write data to tables in the database is the same for any DBMS:

- 1 Establish a connection to the database.
- 2 Send a correctly formatted SQL statement to be executed on the DBMS server.
- 3 Process the results.
- 4 Close the connection.

Establishing a connection to the database

One of the advantages of using a database is the mandatory security built into the DBMS server. Before you can establish a connection to a database, you need to create a user account through which your application will connect. By default, the DBMS server will be installed with an administrator level access user account (usually named the root account). Most application developers will create a single user account to interact with the database on behalf of the application but it is also possible to set up user accounts for individual users and establish user specific connections. Database security can be controlled through the administration console of the DBMS or through the SQL statements CREATE USER, CREATE ROLE, GRANT and REVOKE.

For example, to set up application access to a database named **myDB** for a user named **myAppUser** with password **XQ8d78yB4s** in standard SQL, which should be supported by most database servers, use:

```
GRANT CONNECT TO myAppUser IDENTIFIED BY XQ8d78yB4s
```

In Oracle and MySQL, use:

```
CREATE USER myAppUser IDENTIFIED BY XQ8d78yB4s
```

In PostgreSQL, use:

```
CREATE USER myAppUser WITH PASSWORD 'XQ8d78yB4s'
```

In SQLServer, use:

```
CREATE USER myAppUser WITHOUT LOGIN
```

Note that SQLServer also allows users to be mapped to an active directory login. This allows the user to connect to the database, but it is also important to grant security privileges to the application user to permit specific types of SQL statements on specific database tables. During testing, it is acceptable to grant all access to all tables in the database with an SQL statement such as:

```
GRANT ALL ON myDB TO myAppUser
```

Eventually, before publishing the digital solution for wider access, it would be important to review the database security and revoke any privileges that are not required – for example, to revoke the DELETE privilege for the **usrTable** table in the **myDB** database:

```
REVOKE DELETE ON myDB.usrTable FROM myAppUser
```




3.3.6 SQL tutorial

Once you have configured the privileges for the application user account within the DBMS administration interface, it is helpful to write a reusable code segment that can be included in any parts of your coded solution that need to interact with the database. This code segment will include the function calls to connect to the database server as the application user you have configured, and will select the application database to use.

The syntax for the steps required to establish this connection with the database server will be documented in the API for the programming language and DBMS server you have selected. An example of what this code might look like in PHP and MySQL is:

```
<?php
$my_server="127.0.0.1"
$my_user="myAppUser"
$my_password=" XQ8d78yB4s"
$my_db=" myDB"
//Open a connection to the database
$dbconn = mysqli_connect($my_server, $my_user, $my_password, $my_db);
//Report connection status
if ($dbconn) {

    echo "Success: ".$my_user." is connected to ".$my_db;
} else {
    echo "Error: Unable to connect to database. ";
    echo "SQL Error: ".mysqli_connect_error();
}
?>
```

If you are using a single-user DBMS server such as the SQLite Python library, then the security setup will not be required as the database is not accessible to, and does not require protection from, outside connections. In Python SQLite, the database connect code would look like this:

```
import sqlite3
def getDbCursor(db_file):
    try:
        conn = sqlite3.connect(db_file)
        return conn.cursor()
    except sqlite3.Error as e:
        sqlError='SQL Error: '+e
        print(sqlError)
    return None
dbconn=getDbCursor ('my_db.db')
```

ACTIVITY 3.3.4



Generate



Evaluate

- 1 Using the DBMS you have selected for your Unit 3 project, create a simple test database that contains one table with four columns of data. The code samples will refer to the table as PlayerData and the columns as name, gender, favColour and highScore, but you should use meaningful names and adapt the code to match your column names.
- 2 If appropriate, use the administration console to create a test user account and use the SQL interface of the administration console to grant the test user access to the test database.
- 3 Find an online reference to the API for the programming language and DBMS server you have selected for your technology context.
- 4 Create code samples such as the ones above using the syntax specified in the API.
- 5 Test and debug your code.

Sending an SQL request

Once the connection to the database has been established, it is possible to send any SQL statement to the database, just as you would if you were using an interactive SQL interface. In an interactive SQL interface, you would type the SQL statement into a window and click a button to execute it. In a coded interface, you store the SQL statement in a variable and then send it to the server as a parameter to a function.

For example, to select all the records from a table named `playerData`, you would use the following SQL statement:

```
SELECT * FROM playerData;
```

To send this request to a MySQL database server using PHP code, you would use code like this:

```
<?php
//construct the SQL query statement
$sql="SELECT * FROM playerData;";
//execute the query
$result=mysqli_query($dbconn, $sql);
//check success or failure
if ($result) {
    $num_rows=mysqli_num_rows($result);
    echo("Success: Your query returned ".$num_rows." rows.");
} else {
    echo "Error: Unable to execute query. ";
    echo "SQL Error: ".mysqli_connect_error();
}
?>
```

Sending the select player request to an SQLite database server using Python code would look like this:

```
try:
    sqlStmt='SELECT * FROM playerData'
    dbconn.execute(sqlStmt)
except sqlite3.Error as e:
    sqlError='SQL Error: '+e
    print(sqlError)
```

This process is usually the same regardless of whether data is being selected, updated, inserted or deleted. For example, in PHP MySQL, replacing the `SELECT` statement with an `UPDATE` statement would require no change to the rest of the code except that `$num_rows` would be replaced with `$num_affected_rows`.

Processing SQL results

The result from a coded SQL query is usually returned as a multi-row dataset accessible from the variable used to store the function results. The data structure of the result set is often in a specialised format, so the API provides an SQL fetch function to allow the developer to access the result dataset row by row. The most common approach is to cycle through the result dataset row by row until the end of the dataset is reached. If the exact number of rows in the dataset is known in advance, a counted loop such as `FOR` can be used. In many cases, unless an SQL `COUNT` or `num_rows` function has been used, the exact number of rows may not be known. The most popular control structure for traversing a results dataset is a pre-tested loop such as `WHILE`.

ACTIVITY 3.3.5

- 1 Extend the program you created in the previous activity.
- 2 Create code samples to select the data from your test table using the syntax specified in your API.
- 3 Test and debug your code.



Generate



Evaluate



In PHP/MySQL, the code to iterate through a results set would look like this:

```
<?php
// Fetch all records into an associative array
while ($row = mysqli_fetch_assoc($result)) {
    // Use a formatted print string to print each row
    printf("Name: %s | Gender: %s | Favourite Colour: %s | High Score:
    %s", $row['name'], $row['gender'], $row['favColour'], $row['highScore']);
}
// Release the memory used to temporarily store the results
mysqli_free_result($result);
?>
```

In Python SQLite, the code to display the rows in the dataset would look like this:

```
rows=dbconn.fetchall()
for row in rows:
    print(row)
```

ACTIVITY 3.3.6



Generate



Evaluate

- 1 Extend the program you created in the previous activity.
- 2 Create code samples to display the data from your test table by iterating through the results set.
- 3 Test and debug your code.

Close the connection

It is good practice to close the database connection once the results table has been processed.

In PHP MySQL, this would look like this:

```
<?php
mysqli_close($dbconn);
?>
```

In Python SQLite, it would look like this:

```
dbconn.close()
```

ACTIVITY 3.3.7



Generate



Evaluate

- 1 Complete the program you created in the previous activity by adding code to close the database connection.
- 2 Test and debug your finished program.

Coding for file input/output

A file is a collection of data stored on a persistent storage device such as a disk drive. The format of a file will depend on the operating system platform, character encoding format or application-specific format indicated by a special file extension. Some files are extremely small and can be loaded into memory in entirety; other files are very, very large and need to be managed with a buffering system. All programming languages have some way of interacting with persistent files, whether it is part of the native code, an imported module or a library with specific file-handling features. While every programming language has its own unique way of interacting with files, there are some common concepts and processes that are relevant for code that needs to create, read, write, open and close a file. When working with any file, the process will usually be:

- 1 Open the file with a specific type of access mode.
- 2 Read or write one or more lines of the file to/from a memory buffer.
- 3 Repeat step 2 as many times as necessary.
- 4 Close the file.

File access

Before you can open a file, you need to know where it is. The file path is the location of the file relative to the current operating system directory, and is not necessarily the same folder as the running program. Whether accessing an existing file or creating a new one, it is important to know where the file will be stored. Some operating systems allow a system environment variable to be established with a list of possible directories that may contain required files, but in most cases it is important to know not just the operating system filename, but also the directory path to the file. The location of a file may change regularly, so it is helpful to use a variable to hold the path of the file location somewhere towards the beginning of the program. That way, if the location of the file changes, it can be easily updated in one place, and this reduces the number of edits required throughout the code. As a first step, it is also important to check that the file exists.

For example, in Python:

```
import os.path
fname='/data/datafile.txt'
if os.path.isfile(fname):
    print("File exists")
else:
    print("Can't find file")
```

Or in PHP:

```
<?php
$fname = '/data/datafile.txt';
if (file_exists($fname)) {
    echo "The file $fname exists";
} else {
    echo "The file $fname does not exist";
}
?>
```

Once you have established that the file exists, you need to decide what type of access mode you need to use to open it. Most programming languages have a range of different access modes, such as read-only, append, and read/write. Opening a file in read-only mode allows other programs to read it at the same time, but not write to it. Opening the file to append or write to it will usually create an operating system-level file lock that prevents access from all other programs.

To open a file in read-only mode in Python:

```
myfile = open(fname, "r")
```

And in PHP:

```
<php?
$myfile = fopen(fname, "r");
?>
```

In both of these examples, the 'r' or read access mode is used. The file will be opened for reading only, and the file pointer, which 'points' to information about the file, will be placed at the beginning of the file.

'r+' or 'rw' is used to indicate the file is opened for both reading and writing, and the file pointer is placed at the beginning of the file.

'w' or write access is used to open a file for write-only, and the file pointer is placed at the beginning of the file. Opening a file for write access will usually truncate the file and overwrite the contents of the file if it exists. This is often used to create a new file.

'a' or append access will open a file for writing and place the file pointer at the end of the file. New content is then appended to the end of the file.



'c' is less common, but is usually used to create a new file. If the file already exists, the new file is treated in a different way depending on the programming language. Sometimes the file will be truncated and the contents overwritten, and sometimes the existing contents of the file are preserved. The file pointer is usually placed at the start of the file.

ACTIVITY 3.3.8



- 1 Find an online manual for your favourite programming language and make a note of the file access modes applicable to that programming language.
- 2 Create a text file using a simple text editor and save it in a subfolder named 'data'.
- 3 Write a program to check whether the file exists.

File structure

File structures are specific to the operating system on which they are stored and the application format used to create the file. The file extension is the best indicator of the possible format of a file. Despite the wide range of file extensions available, most programming languages treat files as one of two types – binary files or text files.

Text files

Text files contain alphanumeric characters enumerated into binary codes using a standard encoding system. They also contain special characters that can be used to indicate the end of the line or file. Text files can be read all at once, or line by line, using the end-of-line (EOL) marker to determine the end of each line. The end of the file is marked by the EOF marker.

The operating system determines the default encoding system used to enumerate text into binary codes. Most systems use the Unicode UTF-8 encoding format, but some operating systems use proprietary encoding or non-standard end-of-line markers. If not specified by the program, the encoding system used will be the default encoding format for the current operating system. File open functions often have an option to specify an alternate encoding system to allow for cross-platform compatibility, or a special flag or function to convert end-of-line markers from one system to another.

For example, MAC OS X and Linux use a single line feed (LF) character to indicate the end of a line, but Windows uses a carriage return (CR) and line feed (LF) character to do the same thing. Older Apple systems use a single carriage return (CR) character without LF. These variations can result in frustrating errors when writing files that need to be accessed by multiple operating systems. Some Windows implementations of programming languages offer a 't' (text-mode translation) flag to allow text files created on non-Windows systems to be interpreted correctly.

Binary files

Binary files such as images and video files contain a stream of binary numbers formatted according to the encoding system used for the specific file type indicated by the file extension. These files need to be opened in binary mode (usually by adding a 'b' to the end of the file access code). Binary files can only be read as an input stream as they do not contain EOL markers.

Most files will have a file header. The file header is usually a few bytes at the beginning of the file reserved for metadata about the file. The unique combination of binary codes at the beginning of the file header is sometimes referred to as the magic number, and can also be used to guess the format of a binary file.

Reading a text file

The expected size of the file often determines the best way to access the file contents. If the file is small and the available memory is large, it is easiest just to read the entire contents of the file into memory.

For example, to load an entire small file into memory in Python requires the code:

```
lines = myfile.readlines()
```

In PHP, this can be done with the code:

```
<?php
$lines = file(fname);
?>
```

However, if the file is large, this approach may not be possible. In this case the file needs to be traversed line by line, using a loop control structure, in a similar way to the result dataset of a database.

For example, to read a large file line by line and echo the results in Python:

```
# Read the first line
line = myfile.readline()
# If the line is not empty print the line and read the next line
# keep doing this until the EOF marker is reached
while line:
    print line
    line = myfile.readline()
```

And in PHP:

```
<?php
while ($line = fgets($myfile)) {
    echo($line);
}
?>
```

If the text file is in a delimited format such as CSV (Comma Separated Values), there are usually specialised libraries that make it easy to split each line into fields.

For example, in Python:

```
import csv
with open(fname, "r") as csvfile:
    readcsv = csv.reader(csvfile, delimiter=",")
    for row in readcsv:
        firstValue = row[0]
        secondValue = row[1]
        # do something with firstValue and secondValue
```

And in PHP:

```
<?php
// open the file as you would for any other text file
$myfile = fopen("data.csv", "r");
//use a special csv read to treat each line as an array of values
while ($dataValues = fgetcsv($myfile)) {
    $firstValue = $dataValues[0] ;
    $secondValue = $dataValues[1] ;
    // do something with firstValue and secondValue
}
?>
```

ACTIVITY 3.3.9

- 1 Update the program you started in the previous activity to read your small test file line by line and display it on the screen.
- 2 Test the program works with your small test file.
- 3 Update the filename variable to reference a larger file in a different location.
- 4 Test your program works with the larger file.



Generate



Evaluate

Reading and writing binary files

Reading and writing binary data is done byte by byte, rather than line by line. Images, sound files and video files are all stored as binary data files.



For example, JPEG files always use a specific file format. The first three bytes of a JPEG file are always hexadecimal FF D8 FF. To check if a file is a JPEG image in Python:

```
import binascii
binary_file = '/images/myPic '
#convert the hexadecimal values to binary format
jpeg_sig = binascii.unhexlify('FFD8FF')

with open(binary_file, 'rb') as file :
    #read the first three bytes of the binary file
    first_three_bytes = file.read(3)
    # compare the bytes read to the expected jpeg signature
    if first_three_bytes == jpeg_sig:
        print("Filetype: JPEG")
    else:
        print("Filetype: unknown")
```

And in PHP:

```
$binary_file = '/images/myPic ';
//convert the hexadecimal value to binary
$jpeg_sig = pack('H*', 'FFD8FF');
$myfile = fopen($binary_file, "rb");
//read the first three bytes of the binary file
$first3bytes = fread($myfile, 3);
//compare the bytes read to the expected jpeg signature
if ($first3bytes == $jpeg_sig) {
    echo("Filetype: JPEG");
} else {
    echo("Filetype: unknown"); }
```

Data streams from IoT devices are also stored as binary files. In this case, a file buffer in memory is used to capture the input from a data stream and the buffer is regularly written to disk several bytes at a time. For example, maybe you need to read serial data from an accelerometer controlled by an Arduino, and save it to a binary file. In Python, this might look like:

```
import serial
fname='accel.dat'
with serial.Serial('COM4', 9600) as port, open(fname, 'ab') as binary_file:
    for i in range(0, 10):
        x = port.read(1)
        print x
        #write the value to the file buffer
        binary_file.write(x)
        #write the buffer to the disk
        binary_file.flush()
```

When recording streams of data, it is important to understand that writes to memory will be much faster than writes to disk. If the data logger program starts missing packets because the data stream frequency is high, the data may need to be buffered in memory for a while and regularly flushed to disk before the memory buffer fills up.

Cleaning and merging data

Data is valuable and it is expensive to collect. Organisations are recognising that one big dataset can be used by multiple applications. However, an existing dataset that has already been collected will not necessarily conform to the data requirements of the new application. The complicated process of repurposing, reformatting and combining large datasets from multiple sources is becoming an essential skill in the fields of

business analytics, information technology and data science. While the datasets you will use in your projects are not nearly as large as those used in these fields, understanding the process of cleaning and merging data is beneficial when generating a digital solution.

The process that evolves multiple disparate datasets into one clean, merged dataset is an iterative, non-linear process that is more accurately represented in figure 3.3.11.

Cleaning or scrubbing data

Cleaning or scrubbing data is a necessary first step before merging data. This ensures that the data is free of errors and inconsistencies that could cause problems when datasets are combined. For example, some datasets will use spaces or empty fields to indicate that data is missing, and other datasets will use a default value such as 9999 to indicate data fields that are missing or outside the required range. There may also be formatting variations between different datasets, such as storing gender values as M and F or 1 and 2. Names may be stored as a single field in one dataset, and separately as first name and last name in another. If two datasets need to

be combined or filtered, the key fields used to filter or combine the data in each dataset need to be identical. A human may be able to tell that John SMITH is the same as Smith, John, but a computer requires fields to contain identical data representations if it is to recognise that both fields are equal.

Common data errors that need to be cleaned include:

- **Mismatched data representations**

Queensland and QLD are the same to a human, but not to a computer. To a computer, the street name Johnson Ave is not the same as Johnson Avenue.

- **Duplicate data**

Duplicate data could be intentional or caused by human error. For example, someone might create a new account each time they forget their password, creating duplicate customer records. Multiple tenants at a rental property may be listed as separate records. How do you know if those records are of people who are sharing the property now or of past tenants?

- **Summary data**

Datasets are often published with headings, footers, and summary lines containing counts, averages and totals. These lines should be removed before the datasets are merged.

- **Redundant data**

A typing error in a key field or an error in the data-collection process may result in multiple records for the same entity. Some large read-only datasets intentionally include redundant data to improve speed of access. This type of redundant data needs to be stripped from the dataset before a merge.

- **Categorised data**

One dataset may categorise people as youths aged 14–18 or adults over 18, and another dataset may contain their specific age. Temperature condition labels such as warm and cool may be used to categorise environmental data, rather than the specific temperature recording. In order to successfully merge these datasets, a common category needs to be established.

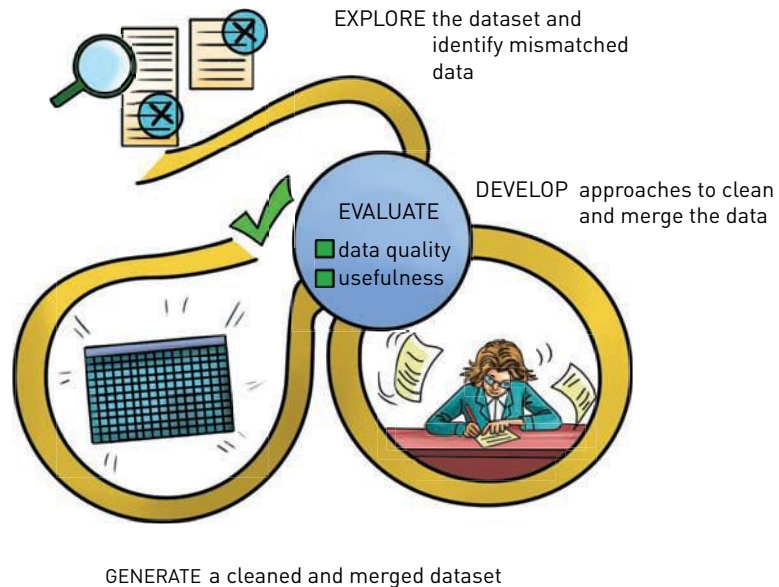


Figure 3.3.11 The problem-solving process for cleaning and merging datasets



Develop approaches to clean and merge the datasets.



Explore the dataset and identify disparate data formats.



Generate a cleaned and merged view or copy of the data.



Evaluate the data quality and usefulness of the data.

- **Spelling errors**

Spelling errors are usually a problem with longer text fields. Spelling errors in a key field should be resolved as a high priority as they can lead to duplicate or mismatched data. Spelling errors in other fields can cause embarrassment when data is displayed in the final solution. It is best to remove spelling errors using spellcheck facilities at some stage, but it is not critical to resolve all spelling errors before merging datasets.

Exploring the datasets

Publicly available datasets are usually published as CSV or XML files and could be presented in tabular, hierarchical or network structures. Privately owned database files can usually be exported as SQL DDL statements or CSV files.

The most popular dataset format is tabular, in which data is organised into rows and columns in a machine-readable CSV format. A good way to explore tabular data is to import the dataset into a spreadsheet or database format to view the data and how it is formatted. Spreadsheet pivot tables and SQL `SELECT DISTINCT` statements can be used to look at the format of specific columns and the way data fields are represented.

Geospatial data is more complicated and may be represented in a hierarchical model organised by country and state, or a network model organised by roads. Geospatial data is usually represented in an XML or KML format. Using spreadsheet tools to access geospatial datasets is possible but some of the hierarchical information is lost.

ACTIVITY 3.3.10

Visit the European Data Portal and review the information there about common data errors.

- 1 Complete the quiz on the web page to check your understanding.
- 2 Visit the Open Data Institute website and explore the different approaches to removing these common data errors.

Developing a cleaning strategy

There are a few different approaches to cleaning data.

- If the data is in a CSV format, create a new copy of one of the datasets to be merged and write a program that reads the dataset line by line, updating or deleting disparate data.
- Import the dataset into a database format. Write SQL `DELETE` statements to remove or update anomalous data from the dataset.
- Use a specialist data-cleaning program that allows common rules to be configured to automatically clean or repair data in each dataset.
- Write algorithms that use a regular expression pattern to search for known variations of field formats, and ignore or replace the data with a consistent equivalent format during the merge process.

Many of these approaches assume that the data source is in a flat file format. It is also possible to combine data from multiple databases. When combining multiple databases into a single database it is important to ensure that the domain constraints for each database definition are consistent. For example, if the database specifications for the digital solution dictate a length of 32 characters for a field and the corresponding data in the data source is 128 characters, it will be necessary to review whether the digital solution can be adapted for a larger sized field or if truncating the data is necessary.

During the process of exploring the dataset and developing cleaning approaches, it is worth periodically evaluating whether the process or algorithm considered to clean the data is time-efficient and whether it complies with the data requirements for the digital solution. It is important to minimise the need for manual inspection and correction of data and to document and automate as much of the cleaning process as possible. This way, the process can be repeated and reused in other datasets, or adjusted and repeated if anything is missed in the first pass. Once a prototype of the digital solution is available for testing, additional errors may be discovered. If the errors identified are likely to cause problems for other users of a shared dataset, it is good practice to advise the data owner of any spelling errors or data inconsistencies you find so that the data owners are aware of the problems and can improve the dataset for use by other developers.



3.3.7
Viewdata



Explore



3.3.8 How to
clean your data

Merging datasets

The purpose of merging datasets in this unit is to combine two different collections of data to create new information that can be used in an innovative digital solution. It is likely that one or more of the datasets to be merged has been collected by a third party and contains some information that is not required by your solution. Merging datasets involves excluding unnecessary data and identifying relationships between fields.

The rules required to merge two datasets can be defined using SQL. Data from two or more tables can be combined using an SQL statement with a JOIN clause. If the join process is complex and involves intermediate aggregation of the data, it may be necessary to use the SQL SELECT INTO statement to create a temporary table as an intermediate step. Even if the data is in a flat file format, SQL statements can accurately specify the rules to combine two or more datasets in a tabular format. It is not always necessary to physically merge the datasets. Combining smaller datasets can be achieved with SQL join statements or coded merge functions as an additional step during execution of the program. Some programming languages and DBMS implementations include specialised MERGE commands that make it efficient to clean and merge sorted data in a single step.

PROCESS FOR MERGING DATA

The process that should be used to merge two data sources is as follows:

- 1 Document the table schema of each data source.
- 2 Document the table schema for the target database.
- 3 Identify the relationships between source and target schemas.
- 4 Define the merge rules using SQL.
- 5 Merge the data sources using SQL or SQL-like coding libraries.
- 6 Evaluate the success of the merge, adapt and redo if necessary.

Documenting the table schemas

A table schema is the structure of a table. The information that should be recorded for each table schema includes the name of the field, the data type and any additional information that might be helpful to know during the merge. This can be represented as a UML diagram, a relational schema format or as a simple table, as shown in table 3.3.5.

Table 3.3.5 Sample table schema

AUSTRALIAN ATHLETES				
Field name	ID	Name	Gender	DOB
Data type	INT	VARCHAR(50)	INT	DATE
Additional information	Primary Key	Combination of first name and surname	1 = Male 2 = Female 0 = Not specified	dd/mm/yyyy

USA ATHLETES				
Field name	ID	Name	Gender	DOB
Datatype	INT	VARCHAR(50)	INT	VARCHAR(10)
Additional information	Primary Key	Combination of first name and surname	1 = Male 2 = Female Null = Not specified	mm/dd/yyyy

COUNTRIES				
Field name	Country	Flag Colours	National flower	Commonwealth
Data type	VARCHAR(50)	VARCHAR(30)	VARCHAR(40)	BOOLEAN
Additional information	No abbreviations used.	Multiple colours are separated by commas		True if the country is a member of the Commonwealth



EVENTS				
Fieldname	Event	Sport	Location	Type of Event
Data type	VARCHAR(256)	VARCHAR(50)	VARCHAR(50)	INT
Additional information				Ind. Men=1 Ind. Women = 2 Team Men =3 Team Women = 4

RECORDS				
Fieldname	Event	Name	Year	Time
Datatype	VARCHAR(30)	VARCHAR(30)	VARCHAR(4)	VARCHAR(20)
Additional information				

Identifying the data relationships

Not all datasets are related. If there are no common columns between the datasets, there is no relationship and they cannot be merged. If there are no common rows between datasets and several common columns, it may be possible to simply append or concatenate one dataset to another, removing columns that are not common to both. In this case an SQL INSERT INTO statement would be more appropriate than a join. In all other situations, the merge should be represented as an SQL join statement. However, before attempting to merge two data sources we must first identify the type of relationship between them. There are four types of relationships: one-to-one, one-to-many, many-to-one and many-to-many.

A one-to-one relationship between two datasets occurs when the primary key column is common to both. Each row in one table matches a row in the other. For example, if you wanted to add a world record time field to the events dataset in table 3.3.5, you would need to match the 256-character event name in the Events data set to the equivalent 30-character event name in the Records data. The world record time could then be added to each appropriate row in the Events dataset. To simplify this example, assume that the data-cleaning process has updated the Event column in the events table to match the abbreviated format used in the Records table. The SQL join for this merge would look something like this:

```
SELECT events.*, records.time
FROM events, records
WHERE events.event = records.event
```

A one-to-many or many-to-one relationship between two datasets occurs when a primary key field in one table appears many times in the other table. In this situation, merging the two datasets will create redundant data, and a physical merge may not be necessary as it will create a table that is not in third optimal normal form. The exception to this rule is that, if there are rows that appear in one table that do not appear in the other, it may be worthwhile to merge the data sources to assimilate the data and then complete a normalisation process on the combined data table. In the Commonwealth Games example shown in table 3.3.5, a developer may wish to combine data from the Commonwealth Games event schedule with the data for individual athletes in order to create a personalised timetable for each athlete:

```
EventSchedule=[EventID, Date, Time, Location]
EventParticipants=[EventID, AthleteID]

SELECT AthleteID, Date, Time, Location, EventSchedule.EventID
FROM EventSchedule, EventParticipants
WHERE EventSchedule.EventID = EventParticipants.EventID
SORT BY AthleteID, Date, Time
```

A many-to-many relationship between two datasets occurs when a field appears multiple times in both tables. It is not usually necessary to physically merge datasets with a many-to-many relationship, but it is sometimes helpful to join the data in many-to-many relationships with an SQL join. The most

effective way to join tables with a many-to-many relationship is to convert the many-to-many relationship into a many-to-one and a one-to-many relationship. This is done by creating a third table that maps the relationships between the two entities. For example, an athlete may compete in multiple events, and an event will include multiple athletes. Instead of storing all the details about athletes and events multiple times, an EventParticipants table could be created to include the athlete_ID and event_ID.

Evaluating the success of a merge or join

Combining tables using SQL is a challenging process that can sometimes result in Cartesian joins and unexpected results. Before adding the merge code to your program and applying it to a large dataset, it is beneficial to test the process using smaller test datasets to ensure the results are combined in the way you expect.

Questions to ask to check for common problems during your evaluation include:

- Are there any duplicated records?
- Has the integrity of the data been maintained?
- Does the final dataset include all the required columns and rows?
- Does the final dataset include unnecessary columns or rows?

An evaluation of the athlete timetable example above may identify that, although the data was cleaned correctly and merged correctly and contained no duplicate data, it did not include the timing and location information for relevant team finals because participants in the final are not recorded in the EventParticipants table until results for the preliminary events have been determined. If this feature was required, additional fields would need to be added to the database to allow relevant finals information to be included on individual athlete timetables.

ACTIVITY 3.3.11

- 1 Extend and modify the table schemas in table 3.3.5 to include all the data required to provide the following information for media outlets reporting about Commonwealth Games events.
 - a a list of countries represented in an event
 - b the list of sports represented at each location
 - c the number of athletes representing each country
 - d the date, time and location of all finals events.
- 2 Define SQL joins that will combine the datasets to provide a single results table including all the required data.
- 3 Create test data and evaluate the success of the merge or join.



Generate



Evaluate

TEST YOUR KNOWLEDGE

- 1 List the four principles of the Agile Manifesto.
- 2 List four common types of errors users make when submitting data on a web form or application screen.
- 3 List four strategies for preventing user errors, and one method of providing user feedback.
- 4 List four examples of dirty data. Hint: Dirty data is data that needs to be cleaned.
- 5 Describe an activity that takes place in each of the four phases of the problem-solving process.
- 6 Describe what is meant by the term 'event handler'.
- 7 Describe the four steps that are required to obtain query results from a database using a procedural programming language.
- 8 Describe the process that should be used for reading data from a large text file.
- 9 Describe what is meant by the term 'table schema'.
- 10 Explain what is meant by a user story.
- 11 Explain the main difference between a database and a CSV file.
- 12 Explain how code comments can be used to remove ambiguity in a program function.
- 13 Explain how a look-up strategy can help improve data quality.
- 14 Explain what is meant by 'iterating through a result dataset'.
- 15 Explain the difference between a one-to-one and a one-to-many data relationship.
- 16 Why is it important to adhere to pseudocode conventions when writing algorithms for a programmed component of a digital solution?
- 17 Why is it important to clean a dataset before combining it with another dataset?
- 18 What types of problems are caused by duplicate data when two tables are joined or merged?
- 19 Compare and contrast the properties of text and binary files.



APPLY YOUR KNOWLEDGE

- 1 In a group of between two and four students, use an agile project management process to create a web application that helps councils identify roads that are dangerous in rainy weather. The requirements for the web application are:

MUST	When a location is entered in a web form, the web application displays details of crashes that have occurred during the last 6 months at that location
SHOULD	- When the 'rainy days only' checkbox is ticked, only the crashes that occurred during rainy weather are shown - Populate the location input field on the web form with data from a database
COULD	Include an option for the user to select whether or not the data is displayed in a table, in a graph or as markers on a map
WON'T	Use live data

- a Explore the following datasets:
 - i Queensland crash data from the Queensland Department of Main Roads. You can access crash data from the Queensland Government data website.
 - ii Weather data available from the Bureau of Meteorology.
 - b Sketch a web form and annotate the form with validation strategies that will be used for each field.
 - c Generate and test code to build the form.
 - d Generate and test code to validate the form.
 - e Use the problem-solving process illustrated in figure 3.3.11 to clean and merge the datasets and create a local database that can be used by your web application.
 - f Generate and test code that replaces the location input field with a drop-down list.
 - g Generate and test code that selects data based on web form inputs.
 - h Observe other students using the form and identify possible improvements.
- 2 Use pair programming to create a program module for the avatar selection component of a game about famous people. The prescribed requirements are:

MUST	- Accept an occupation name as input - Return a list of names of famous people with that occupation
SHOULD	- Use SQL to select names from the Pantheon dataset* where occupation matches the user input - Use a drop-down list to ensure users only select occupations that exist in the dataset - Return a second list containing the nationality of each famous person
COULD	- Order names by popularity - Limit results to a maximum of 10 names - Display a flag icon to represent the nationality of each person*
WON'T	Display an image of the famous person

- 3 Create a mobile application that allows users to enter a drink product and display the nutritional facts about that drink. The prescribed requirements are:

MUST	- Allow a user to select a drink product - Display nutritional information about the drink product selected
SHOULD	- Display a suitable message if there is no data about the drink product entered - Use a local database such as SQLite populated with open source data**
COULD	- Accept input using a barcode scanner - Display a picture of the product with the nutritional information - Categorise the product as healthy or unhealthy based on nutritional information
WON'T	Include food products

- 4 Create an intelligent system that collects and presents data collected from sensors located throughout the school.

MUST	Log data from sensors in a file or database
SHOULD	- Aggregate sensor data against a timeline from a second data source (e.g. school timetable events) - Include several days of data recordings
COULD	- Present the information in a graphical format - Allow user selection to identify the start and end times for the display
WON'T	Use live data feeds



3.3.9 Crash data
3.3.10 Climate data

3.4



EVALUATING A DIGITAL SOLUTION

In this section, you will analyse test data and user experiences to evaluate the quality and completeness of the solution and make recommendations based on specific criteria.



Evaluating the solution

Evaluation occurs throughout the problem-solving process. Evaluation occurs when we select one idea out of several during brainstorming about each aspect of the digital solution, or apply systems thinking to identify and evaluate potential personal, social and economic impacts such as data privacy and digital copyright. Feedback from peers and users about early ideas is evaluated to refine requirements. Sketches or paper prototypes of the solution are used to evaluate the user experience. As user interfaces, data structures and code are generated and prototype solutions are tested, evaluation of these components detects errors and recommends improvements. After each iteration of an agile development phase, evaluation against the initial criteria occurs to provide feedback for the next development cycle. Finally, at the end of a project, self-determined criteria are used to evaluate whether the goals of the developer have been met.

Establishing evaluation criteria

Evaluation criteria for a digital solution will include prescribed criteria of the solution, determined by functional requirements and technical constraints. They will also include self-determined criteria based on non-functional requirements, programming conventions and developer goals. The criteria for a digital solution may be weighted equally, or prioritised on the basis of their importance to the client. As the requirements of the solution change, so must the evaluation criteria.

Testing the solution

Some aspects of an evaluation can be subjective, but it is important to develop quantitative test data and user feedback that can be used to support judgements and justify recommendations.

Testing functional requirements have been met

A test scenario identifies a general action that can be used to check if a functional requirement is met. A test scenario should be outlined for each functional requirement, describing how evidence can be collected to show that the requirement has been met. The test scenario may include multiple test cases that each outline a more specific goal, such as entering valid data, invalid data or unexpected data. When specifying test cases, it is beneficial to precisely describe specific testing steps and expected results. This makes it possible for an independent and impartial tester to complete the test.

For example, to test the login functionality of an application:

- **Functional requirement**
Users must log in.
- **Test scenario**
A user can log in successfully.

- Test case 1**
When the app is first started, the login screen is presented. Enter the username testuser1. Enter the password TestPassw0rd1. Click the login button.
Test case 1 expected result
A successful login record is created in the application log and the user should be presented with the main menu.
- Test case 2**
When the app is first started, the login screen is presented. Enter the username notausser. Enter the password incorrectpassword. Click the login button.
Test case 2 expected result
No login record is created in the application log and a 'Login unsuccessful' message is displayed in the error message area of the login screen.
- Test case 3**
When the app is first started, the login screen is presented. Enter the userid testuser1. Leave the password field empty. Click the login button.
Test case 3 expected result
An unsuccessful login record is created in the application log and a 'Login unsuccessful' message is displayed in the error message area of the login screen.



ACTIVITY 3.4.1

Create a test scenario and test cases for the following functional requirements:

- Users must be able to update their profile details.
- Email fields must be validated using a double-entry method.
- A student can access their grades for each enrolled subject at the end of each semester.

Once the test cases have been documented, an impartial developer can carry out the testing and record the results. If the actual behaviour of the solution matches the expected result, the test is categorised as passed. If the behaviour does not match the expected result exactly, the test is categorised as failed. The effectiveness of the solution can then be measured based on the percentage of test cases that the solution passed. Screenshots of results can be collected as additional evidence if required.

Each failed test case can then be analysed to determine the cause of the failure and possible changes to correct the problem. After the corrections are made, the test can be repeated as many times as necessary to verify the effectiveness of the modified solution. Only the most recent test results are considered when evaluating the effectiveness of the solution.

Collecting user feedback

User feedback is more difficult to quantify but a rubric such as the example shown in table 3.4.1 can be used to quickly collect impressions about the aesthetics and ease of use of a solution.

Table 3.4.1 A simple rubric can be used to quantify user experiences.

					
I like how the interface looks overall					
I can easily see which buttons to press					
Instructions are clear					
I understand how to use this application					
I like using this application					

Collecting and summarising feedback from a range of users in the target audience can provide quantifiable data about user experiences that can be used to support judgements about the useability, learnability and utility of the solution.

Useability heuristics

Jakob Nielsen's useability heuristics (Nielsen, 1994) can be used as a detailed checklist to evaluate the user interface of a digital solution.

- ☐ **Visibility of system status**
The system should always keep users informed about what is going on, through appropriate feedback and within a reasonable time frame.
- ☐ **Match between system and the real world**
The system should speak the users' language with words, phrases and concepts familiar to the user.
- ☐ **User control and freedom**
Support undo and redo.
- ☐ **Consistency and standards**
Users should not have to wonder whether different words, situations or actions mean the same thing.
- ☐ **Error prevention**
Careful design prevents a problem from occurring in the first place.
- ☐ **Recognition rather than recall**
Reduce memory load by making user choices visible.
- ☐ **Flexibility and efficiency of use**
Provide shortcuts for expert users so the system can cater to both experienced and inexperienced users.
- ☐ **Aesthetic and minimalist design**
Less is more. Keep the design as simple as possible. Do not display data that is rarely needed.
- ☐ **Help users recognise, diagnose and recover from errors**
Error messages should be expressed in plain language, precisely indicate the problem and suggest a solution.
- ☐ **Help and documentation**
Make the system easy to learn but provide help documentation that is easy to search and task-focused.

Source: Adapted with permission from Jakob Nielsen's original 10 heuristics, available at <https://www.nngroup.com/articles/ten-usability-heuristics/>.

Evaluating impacts

In addition to the prescribed criteria identified by stakeholders, it is also important to evaluate potential personal, social and economic impacts such as accessibility, data privacy, intellectual property and digital copyright.

Accessibility guidelines

The concept of accessibility was introduced on page 26. The goal of accessibility compliance is to ensure that a digital solution can be used by the widest range of users possible, regardless of their abilities or disabilities. It is important to evaluate the level of accessibility compliance throughout the development of a digital solution. The accessibility checklist below is based on aspects of Australian, European and US interpretations of the international accessibility standard ISO TC 159, and on additional accessibility toolkits inspired by these standards. The checklist includes accessibility considerations and links to accessibility testing tools for a range of technology contexts. It can be used as a starting point to appraise the accessibility of a digital solution and propose recommendations for further investigation, but the list is not exhaustive. Although there are many automated tools to help developers identify accessibility issues in traditional user interfaces and web interfaces, accessibility guidelines for emerging technologies such as robotics and virtual reality experiences are limited.

- ☐ **Make sure there is enough contrast between text and its background colour.**
Don't indicate important information using colour alone. Assistive technologies read text content, not images and colours.



3.4.1
Colour
contrast
analyser



- ❑ **Don't rely solely on images or sound for understanding and operating content.**
Use text-based labels and positioning data, rather than colours, to communicate geospatial information.
- ❑ **Don't rely on gesture controls or mouse clicks for user interaction; allow users to navigate using a keyboard.**
Users with physical limitations may need to navigate a digital solution using the 'tab' and 'enter' keys. Ensure that the user interface can be configured to allow an appropriate tab order and visible focus.
- ❑ **Use descriptive alt text for images and icons.**
Assistive technologies read the alternative text of an image to describe it to a vision impaired user. Write image descriptions for alt text or image attributes with this in mind so that the user can clearly understand what the image or icon is about.
- ❑ **Provide alternatives for content that cannot be made accessible.**
It may not be practical to make maps and videos accessible, but a video transcript or commentary about the essential elements can be used to communicate the important content.
- ❑ **Use ARIA attributes on web interfaces when applicable.**
ARIA stands for Accessible Rich Internet Applications. It communicates to a web browser how content will change dynamically and what the purpose of the content is. For example, decorative elements can be hidden from a screen reader by adding the attribute `aria-hidden="true"`. The Chrome WAVE extension can help highlight accessibility issues with a web interface.
- ❑ **Give users a way to jump straight to main content.**
A hidden shortcut before the navigation bar, visible only to screen readers, provides visually impaired users a way to jump to the main content without listening to a screen reader announce every menu option it encounters before the main section.
- ❑ **Avoid describing weblinks as 'click here'.**
Assistive technology uses HTML tags to describe web links. It is not very helpful if all the links say 'click here'. Make web links descriptive.
- ❑ **Ensure web interfaces comply with WCAG accessibility guidelines.**
Web tools such as the US web accessibility standards checker and World Wide Web consortium compliance tool can be used to identify parts of a web interface that do not comply with international Web Content Accessibility Guidelines (WCAG).
- ❑ **Consider the needs of vision-impaired users when designing for small screens.**
Small screens such as those on mobile phones are difficult to read. Provide buttons that make a noise when pressed, contrast strongly with the background and have a large click area. Some mobile phones also support large font sizes.
- ❑ **Use accessibility APIs for mobile phone development.**
Most mobile phone companies provide accessibility APIs for mobile app development, outlining specific coding conventions that can be used to improve user interface accessibility through specific attributes and hints. Testing a mobile app using Apple VoiceOver or Google TalkBack (for Android devices) will help evaluate the accessibility of mobile applications.



LEGO Mindstorms robotics programming software is not accessible to blind or visually impaired students. Visually impaired teams that register to compete in the FIRST Lego League are given a special software exemption that allows them to use Notepad++ and NXC with JAWS screen reader software and a braille input device. Making robots accessible to everyone is not just about accommodating impairment. Expensive robotic equipment is financially inaccessible to many student programmers. Many roboticists encourage the use of open-source software such as ROS, low-cost control components such as Arduino boards and Raspberry Pi, and 3D-printed robot components to improve the accessibility of robots. This provides low cost access to robotics equipment to a wider range of developers and the flexibility to create accessible interfaces for intelligent systems.



Data privacy

Anyone collecting personal information about individuals is legally required to obtain consent from the individual to store their information, and must provide opportunities for the individual to update their details or revoke their consent at any time. In order to comply with this legal requirement, applications must provide functionality to comply with the Australian *Privacy Act 1988*. Evaluating a digital solution through the lens of data privacy will help to identify any deficiencies or impacts that relate to the collection and use of personal information.

Personal information is any identifying data that could be used to identify a person. This can include name, date of birth, email address, residential address, or an identifying number related to a workplace or educational institution. Some personal information, such as medical or financial records, is protected by additional laws that relate to the handling of sensitive information.

According to Australian law, when collecting data about individuals, digital solutions must:

- ensure that personal information is only accessed by authorised persons for purposes declared when obtaining individual consent
- ensure that personal information is accurate and kept up to date
- provide users with the option of not identifying themselves by using a pseudonym
- allow an administrator to de-identify or destroy personal information once it is no longer in use
- protect personal information through security controls such as password access and backups
- obtain parental consent before collecting information about a child under the age of 13.

The Privacy Act was amended in 2014 to include 13 Australian Privacy Principles (APPs). A fact sheet about the Australian Privacy Principles is available from the Office of the Australian Information Commissioner. One of the principles relates to cross-border disclosure of information. Many organisations are concerned that they will be held accountable if personal information stored in the cloud is disclosed to a third party. Most cloud storage solutions cannot guarantee that Australian data is held in servers physically located in Australia. Some educational institutions choose not to use cloud services for this reason.



3.4.3
Privacy fact
checker

Intellectual property and digital copyright

During the development of a complex project, it is likely that code samples and modules from other programmers will be used. When you modify and integrate these code samples into your project, it is appropriate to recognise the intellectual property of these silent contributors. The impact of reusing code should be assessed during the project and documented in the final project report. If the digital solution is produced for a competition or commercial venture, formal requests for permission to use third-party code segments may be required.

Similarly, the sources of any images, sounds and videos embedded in a digital solution need to be acknowledged in the method specified by the creator. Some media content providers do not need to be acknowledged, but others require that their ownership is recognised on the user interface display or within the source code of the solution. If the media content is not distributed under a Creative Commons public licence, it may be necessary to contact the author for permission to use the materials. This is particularly important if the solution will be published on the web, submitted to a competition or used for a commercial enterprise. Each media item selected for use in a digital solution should be evaluated to consider the impact of using downloaded media compared with using original media. It is advisable to recommend alternatives in the event that permission to use the media is refused or is too expensive.

Evaluating using self-determined criteria

To create an innovative digital solution, developers often need to learn new tools and approaches. It is helpful to establish personal goals at the beginning of the project and evaluate whether these goals were achieved at the end. Personal goals can include learning, collaborative or productivity targets. Organisational goals may include adhering to organisational project management guidelines, version control and style guides, and producing code that is modular, reusable and maintainable with clear explanations of algorithms in code comments or project documentation.

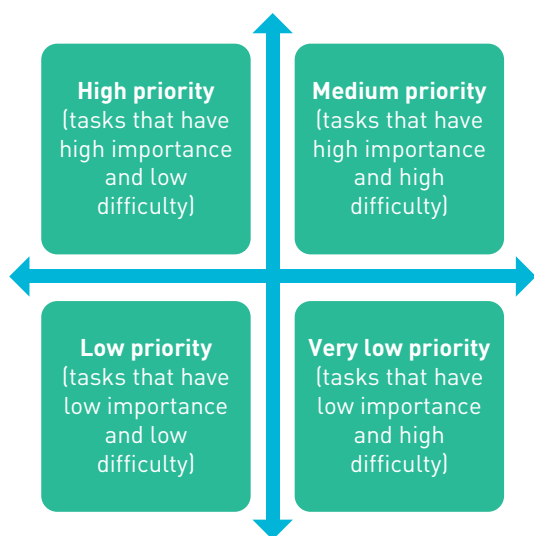


Figure 3.4.1 An action priority approach can be used to prioritise the growing number of improvements identified during the testing of a digital solution.

Evaluating with self-determined criteria will reflect on the reasons why particular approaches were successful for the current project and whether or not they would be successful in similar future projects. It is also important to identify areas for improvement, such as limited access to resources, inefficient project management processes, poor team or client communication and ineffective problem-solving processes.

Justifying and prioritising recommendations

Within the short development cycles common to agile projects, it is likely that time constraints may prevent implementation of many recommendations for improvement. If there are a considerable number of recommendations to implement, it is beneficial to assign an importance and difficulty level to each recommendation as a strategy to prioritise recommendations. An action priority approach such as the one shown in figure 3.4.1 is one possible method of prioritising improvements.

TEST YOUR KNOWLEDGE

- 1 List Nielsen's 10 useability heuristics.
- 2 List four accessibility guidelines.
- 3 Which useability heuristic(s) could be met by including a reset button?
- 4 Which Australian law aims to ensure that personal information about an individual is only collected with their consent?
- 5 Describe the information that should be recorded for a test case.
- 6 Describe a possible impact of using downloaded source code or multimedia assets in your digital solution.
- 7 What is accessibility?
- 8 Explain the difference between a test scenario and a test case.
- 9 Explain how code comments can be used to recognise digital copyright and intellectual property.
- 10 Explain how an action priority approach can be used to prioritise a growing list of change requests for a digital solution.

APPLY YOUR KNOWLEDGE

- 1 Select a few of the projects you have completed during this unit and evaluate them using:
 - a prescribed criteria
 - b self-determined criteria
 - c useability heuristics
 - d compliance with data privacy laws
 - e compliance with digital copyright laws.
- 2 Test and review a project created by one of your classmates. Create a list of possible improvements and explain why the improvement is required.
- 3 Ask a classmate to review one of your projects and recommend improvements. Review the list of improvements suggested, and prioritise them using an action priority approach.
- 4 Individually or within a project team, reflect on the project management approaches you have practised during this unit. Identify three activities or mind-sets that had a positive impact; make a note of these activities so you will remember to continue this in your next project. List two activities or mindsets that had a negative impact on your work, and brainstorm ideas for how to improve in one of these areas.

Unit

4

DIGITAL IMPACTS

INTRODUCTION

Global access to online datasets is an increasing phenomenon that is bringing significant advances in lifestyle, business intelligence, education and social awareness. For example, street directories have been replaced with free online navigation apps, junk mail has been largely replaced by social media advertisements, interactive magazines and newspapers are supplemented with online assets, and open data is used to address social and community problems. The exchange of data across application boundaries opens exciting innovation possibilities, but data privacy and cybersecurity are important considerations that must not be overlooked.

By the end of this unit, students will understand how to create digital solutions that exchange data, consider a range of security strategies to protect data exchanges and evaluate the personal, social and economic impacts of making data available to a global audience.

You will explore and analyse case studies that exchange data to recognise the components of data exchange solutions and investigate data security concepts, network transmission principles and methods of exchanging data over a network.

You will analyse a data exchange problem and investigate possible approaches for data exchange over a network. You will develop data interfaces and algorithms that can be used in data exchange solutions and generate early prototypes of digital solutions that exchange data.

You will investigate a variety of security algorithms and generate coded solutions using a combination of advanced coding structures and code libraries. You will test and refine programmed components to recommend improvements.

You will evaluate the personal, social and economic impacts of sharing data over the Internet. You will use SQL to create summary data suitable for sharing in an open data format. You will use the Digital Solutions problem-solving process to de-identify and encrypt sample data and ensure that privacy principles are upheld.



Explore



Develop



Generate



Evaluate

KEY IDEAS

- Data exchange systems
- Privacy principles
- Data exchange processes
- Data structures
- Methods for exchanging data
- Algorithms
- Data flow relationships
- Problem
- Information
- Solution requirements
- Criteria
- Impacts



4.1

EXPLORING CYBERSECURITY AND DATA EXCHANGE



In this section, you will explore and analyse case studies that exchange data to recognise the components of data exchange solutions and investigate data security concepts, network transmission principles and methods of exchanging data over a network.

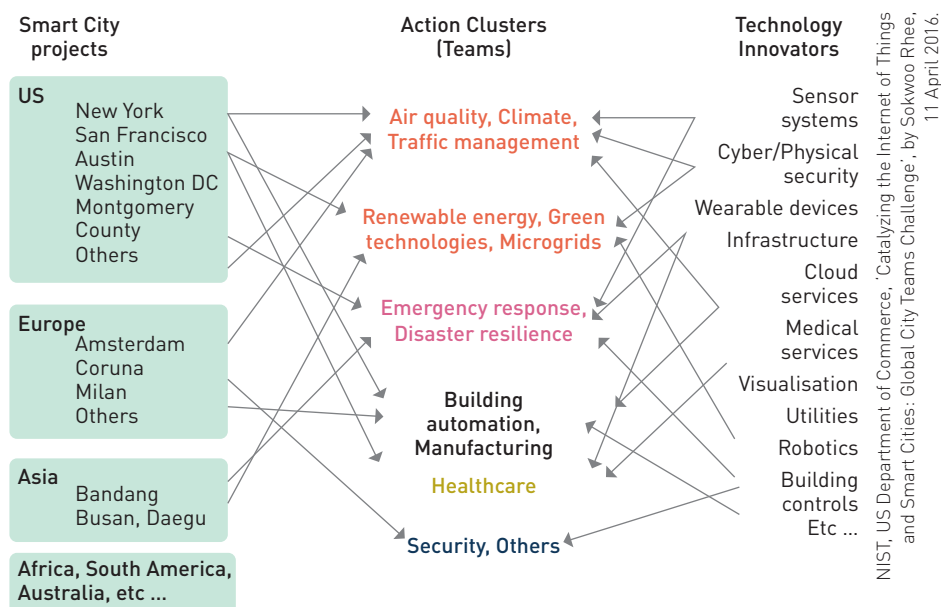


Figure 4.1.1 Many cities share the same challenges but develop custom Smart City IoT solutions. Innovative solutions solve the same type of issue in multiple cities.

The potential benefits to society that emerge when organisations share data are phenomenal. Connected cities projects around the world have shown that using mobile applications, web applications, intelligent systems and interactive media that incorporate live shared data improves the lives of the citizens in the participating cities. For example, Singapore was crowned 'smartest city on Earth' in 2016 because of its successful use of integrated sensor data and an extensive wireless network to address traffic congestion. The Singapore government can also detect whether people are smoking in unauthorised zones, and from 2018, city planners can access a 3D Virtual Singapore model to simulate how people will evacuate in emergencies.



4.1.1 Virtual Singapore

Gartner Group, a well-respected IT research company, predicts that there will be at least 20 billion interconnected IoT devices around the world by 2020. Other, less conservative estimates predict the number will be 50 billion. Due to economic pressures, many of these devices are produced with insufficient RAM to load the security features necessary to protect devices from unauthorised access. Many IT professionals are concerned that these security vulnerabilities open up the possibility of a denial-of-service attack on smart cities of the future. A denial-of-service attack is one in which a large volume of fake network traffic is generated to shut down a network-accessible service. What impact could a coordinated denial-of-service attack have on smart cities of the future?



Data exchange case study

CASE STUDY

OPENACTIVE

OpenActive is a not-for-profit community project funded by Open Data Institute and Sport England. It provides a way for sporting organisations to publish information about classes and fitness sessions in a standard format that can be exchanged with other systems. The goal of OpenActive is to increase participation in physical activities by making information easier to access. Developers can use it to create apps or services that encourage community members to participate in physical activities close to where they are, such as booking a squash court, joining a yoga class or meeting up with a running group.

To make their data available for developers to use, organisations need to create an OpenActive API using a standard programming specification published by the W3C OpenActive Community Group. Once the API is developed, the organisation is added to the Open Active Dashboard (see figure 4.1.3) so that developers can find OpenActive data and incorporate it in their apps.

The website Get Active London is an example of a web application that uses OpenActive data to allow Londoners to search for activity sessions from the different community groups and businesses supplying open activity data. Mobile applications and intelligent systems solutions have also been inspired by the OpenActive data model. Some websites have been set up as both data providers and data consumers to allow smaller organisations to register activity opportunities while also facilitating searches of data. Table tennis clubs and parkour groups are already using OpenActive app servers successfully to promote their activity sessions.



Figure 4.1.2 OpenActive facilitates the exchange of open data to encourage more people to be physically active.

Provider	Endpoint	Uses paging spec?	Uses opportunity model?	Includes coordinates?	Summary	Issues	Licence
activeflowham	✓ No	✓ Yes	✓ Yes	✓ Yes	View	0	Licence
Mubawwat	✗ Open	?	?	✓ Yes	View	5	Licence
Athel	✓ No	✓ Yes	✗ No	✗ No	View	3	Licence
GUL	✓ No	✓ Yes	✓ Yes	✓ Yes	View	6	Licence

Figure 4.1.3 The OpenActive dashboard

Elements of a data exchange system

The two main systems involved in an OpenActive exchange are the data provider system (identified as system 1 in figure 4.1.4) and the data consumer system (identified as system 2 in figure 4.1.4). When the data consumer system requests the data, the data provider system generates a **JavaScript Object Notation (JSON)** dataset and formats it into groups of activities that are transmitted over the Internet to the data consumer system. A standard date format is used to ensure that timestamps can be used reliably to order the data, and to ensure that deleted or out-of-date data is not presented to the requesting system. Dates must comply with the ISO 8601:2004 standard for representation of dates and times – that is, ‘yyyy-mm-ddTHH:MM:SSTZD’. The rectangles around systems 1 and 2 in figure 4.1.4 represent the system boundaries, which will vary for each implementation of the standard by data providers and data consumers.

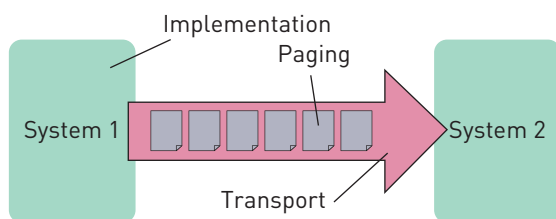


Figure 4.1.4 Elements of the OpenActive data exchange system

The OpenActive REST development architecture is built around an **asynchronous** exchange of open data in real time. In most cases, the provider data already exists in an application database and an API can be created to publish specific data in an open format while still having the flexibility to protect the personal information of its members. The OpenActive API for each activity site responds to requests for open data from a range of data consumers by transmitting relevant activity data generated from the local database.

Determining constraints and requirements of the solution

The main goal of the OpenActive API was to provide a framework that could deliver activity data with speed and simplicity in a flexible and well-documented format. A mind map can be used to explore the requirements, constraints and general considerations for the OpenActive data exchange solution (figure 4.1.5).

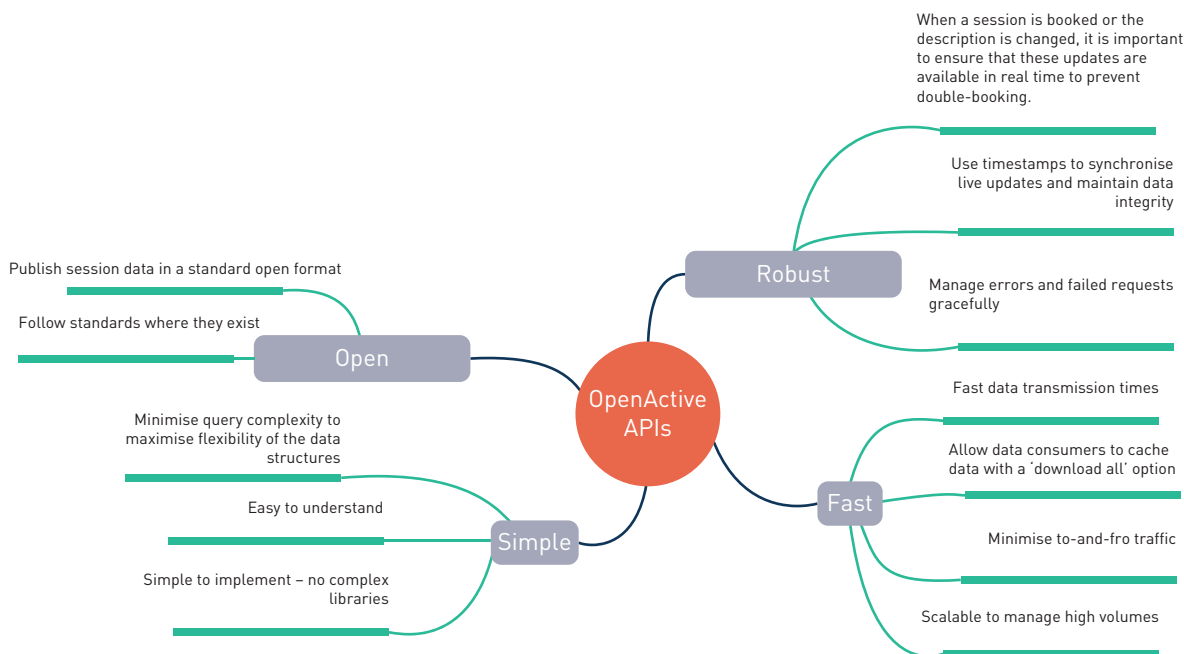


Figure 4.1.5 Brainstorming requirements for the OpenActive API using a mind map

The mind map in figure 4.1.5 identifies the requirements, constraints, considerations and criteria for the data exchange system. These can be represented as a prioritised list of requirements using the MoSCoW method:

- MUST be fast and simple
- MUST allow exchange of data
- SHOULD use timestamps to synchronise updates
- SHOULD allow download of all data
- SHOULD scale to manage peak loads
- COULD support real-time updates
- COULD support unlimited downloads
- WON'T require authentication
- WON'T manage booking and payment
- WON'T guarantee availability of courts and facilities
- WON'T publish membership packages
- WON'T standardise content, e.g. football and soccer will be treated as different categories even though most of the time they will mean the same thing.

Exploring the OpenActive data model

OpenActive data is structured as a physical activity session that identifies a specific event occurring at a specific time and links it to the activity provider or club at a specific location. Description, skill level and cost are additional attributes of the session entity. This can be represented using an ER diagram such as the one in figure 4.1.6.

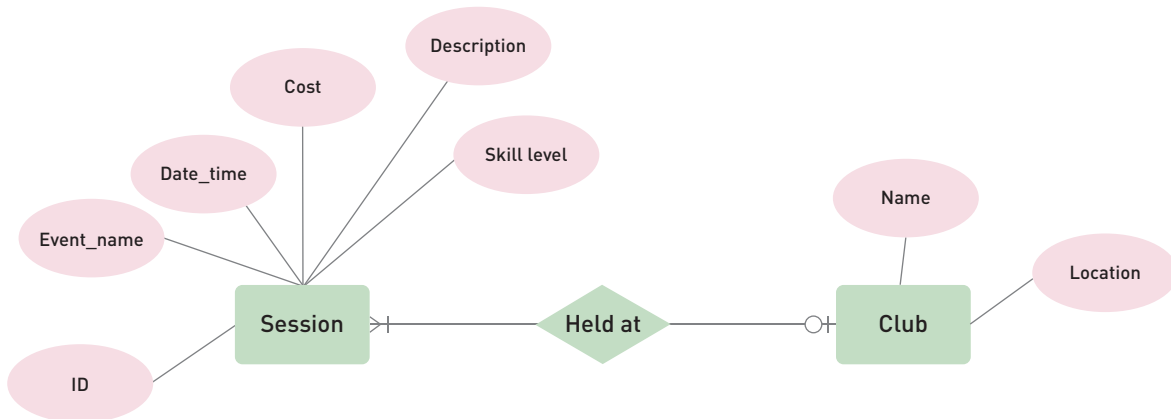


Figure 4.1.6 ER diagram showing key entity relationships in the OpenActive data model

The detailed data model published by the W3C OpenActive Community group uses a Unified Modelling Language (UML) format and is shown in figure 4.1.7.

The data model is populated with data from the application database. Many community sporting groups and businesses have a website presence and a local database to store booking information and membership details. Some of the data in the application database needs to remain private, and some will be published in an open JSON format. This can occur in several ways:

- updating the application logic to update JSON records in memory every time a change is made in the application. The JSON response is generated from the JSON records in memory
- adding columns to the application database to keep track of data required by the API that does not exist in the database. When a JSON request is received, an SQL join is used to gather the data from the application database tables and generate the JSON response
- adding a single de-normalised table to the database that mirrors the required JSON structure, and adding database triggers to keep the new table in sync with updates in the application database. When a JSON request is received, a simple SQL select from the new table is used to generate the JSON response.

The common structure for the JSON data returned in the HTTP response is defined by the OpenActive standard to be:

```

<item> => {
  "state": "updated" | "deleted",
  "kind": "session",
  "id": <id>,
  "modified": <modified>,
  "data": <data>
}
  
```

ACTIVITY 4.1.1

Detailed descriptions of the OpenActive data entities are available at the OpenActive website.

- 1 Read the description of the event data entity at the OpenActive website.
- 2 Set up an empty three-column table with the headings 'Field name', 'Data type', 'Additional information'.
- 3 Populate the table with information from the Events description to create a data dictionary for the Events data entity.



4.1.4
OpenActive

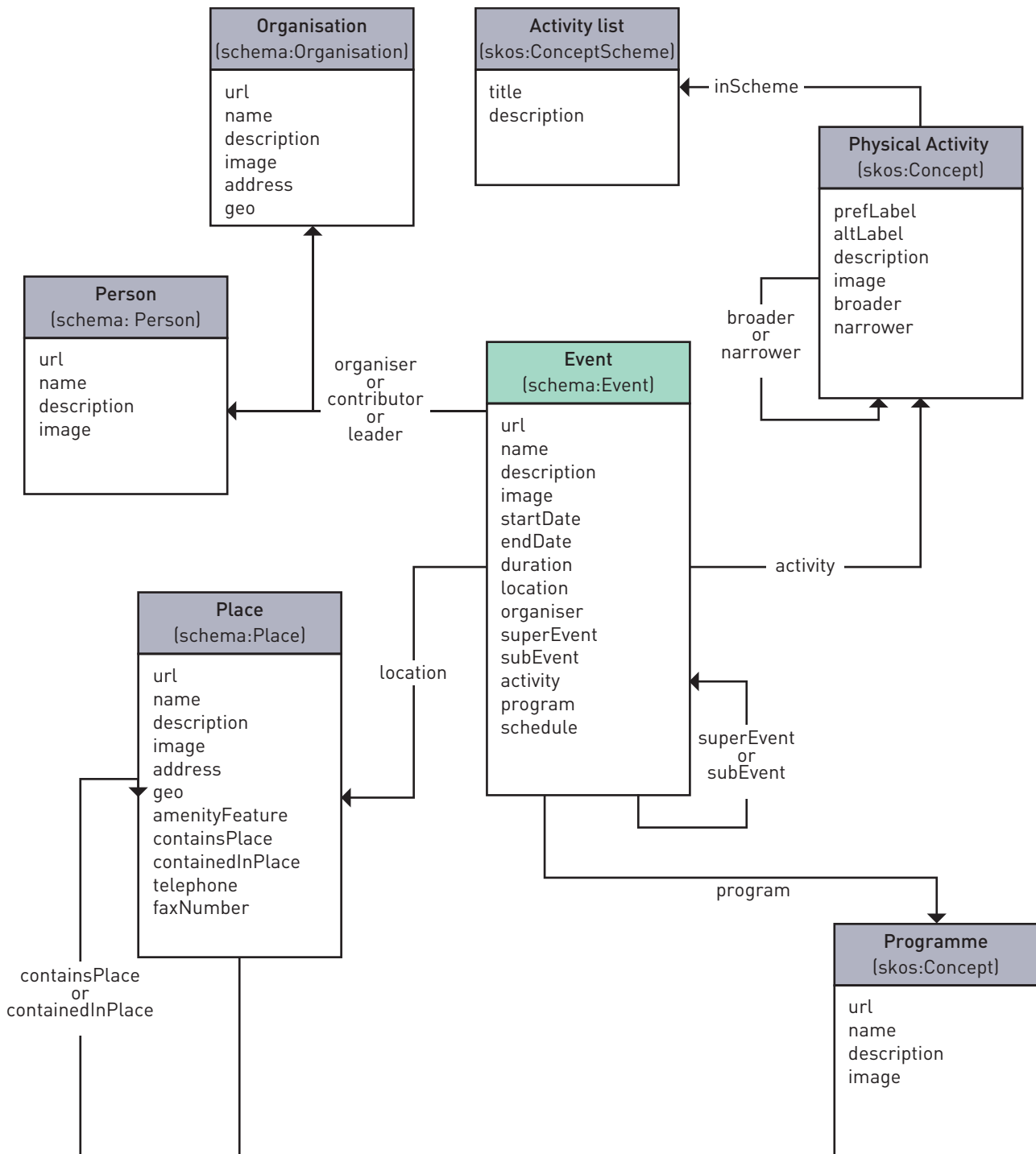


Figure 4.1.7 OpenActive data model

Note: **<modified>** is a timestamp or update sequence number with an integer datatype.

Once all the data is cached, individual records can be requested. If the modified sequence number of the returned data is higher than the cached value, then the entire item is updated.

The **<data>** part of the response is open for applications to populate with whatever data structure is most appropriate for the 'kind' of session data being exchanged. This makes it very open for the data provider but more difficult to combine different data sources at the data consumer end.

Cybersecurity case study

Despite the many benefits that arise from exchanging open data over the Internet, it is also important to consider the impact of unsecured data exchanges. Some types of data, such as personal data, can have devastating consequences if not protected against unauthorised or malicious use. In 2015, personal details of 33 million clients of Ashley Madison, a commercial dating website for married people, were leaked after a malicious hack, and in 2016, a careless developer error during an Australian Red Cross web application upgrade led to the unintentional release of sensitive medical information for half a million Australian blood donors. Cybersecurity is a responsibility enforced under Australian and international law, and it is important to understand developer obligations in relation to protecting data in transit.

In 2014, Australian privacy laws were strengthened to ensure that any privacy breaches were not covered up, but instead were reported immediately, and heavy fines were introduced for non-compliance, to ensure that Australian businesses understood the importance of protecting personal information.

CASE STUDY

AUSTRALIAN RED CROSS BLOOD SERVICE DATA BREACH

The Australian Red Cross collects blood donations for use in medical emergencies. It is an important community service that requires strict controls and detailed screening processes. Some of the personal information collected during blood donation interviews is extremely sensitive and is protected by Australian privacy laws. Any personal or sensitive information disclosed to the Red Cross is required by law to remain confidential. When blood donors enter their personal details on the Australian Red Cross website, donateblood.com.au, they should expect that the information will be transmitted and stored securely.

However, on 25 October 2016, the Red Cross received an alarming call from AusCERT, a cyber-emergency response team based at the University of Queensland, advising them that an anonymous user had discovered a publicly accessible file on the web containing 550 000 blood donor records that included names, dates of birth, contact details and sensitive medical information. The Red Cross immediately shut down the donor site and contacted the Privacy Commissioner to report the breach.

After an extensive investigation, the cause of the problem was identified. Despite strict security controls, an ad hoc backup had been saved to the wrong location during a website upgrade on 5 September 2016. The file had been sitting there, unsecured, for 50 days. Cybersecurity experts immediately took action to investigate how many times the file had been downloaded, and monitored the dark web to determine whether the information was in the hands of cybercriminals. Although it is impossible to be certain, most security experts agreed that it was unlikely that anyone else had downloaded the file.

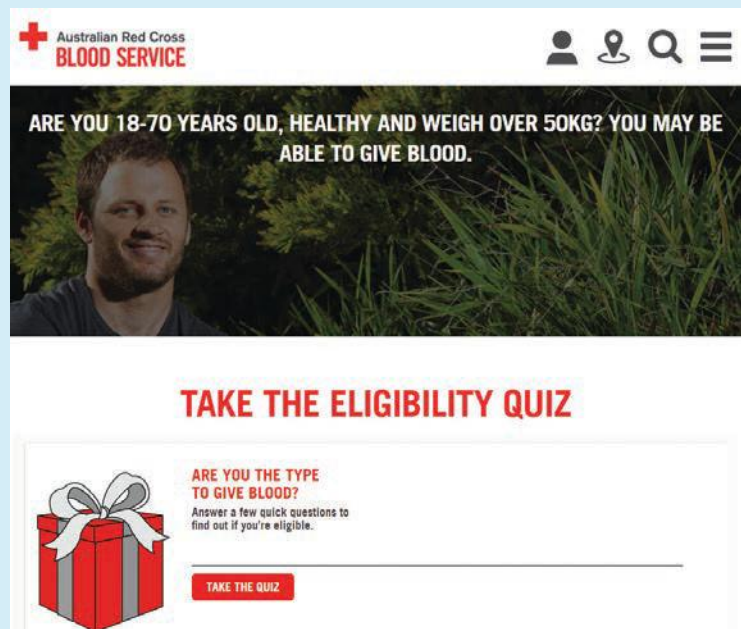


Figure 4.1.8 The Red Cross online donor eligibility assessment collects detailed personal information from potential donors.



Although the impact in this case was minor, the number of records exposed and the detailed personal information that was vulnerable meant that the impacts could have been significantly more serious. Many are labelling the incident the largest online data security breach in Australia's history. The file that was discovered on the web was a complete SQL dump of the test database on a lower security test server. It is not unusual for test servers to have lower security, but it is unusual for test servers to use actual data and for backups to be saved in unsecured areas.

Securing personal information is an important responsibility. A good rule of thumb when collecting personal data is to collect only the details necessary and to protect the data with maximum security. When using data for test purposes, use fictional records, not real data; and open data must always be de-identified so it is impossible to associate an actual person with the data stored.

ACTIVITY 4.1.2



Explore



Evaluate

Follow the weblink provided to find out more about the Australian Red Cross Blood Service data breach that occurred in 2016, and answer the following questions.

- 1 List examples of personal data collected by the Australian Red Cross Blood Service.
- 2 Describe how the security of this data was compromised during the breach.
- 3 Was the data breach accidental or malicious?
- 4 Evaluate the potential impact of the breach to Australian blood donors.
- 5 Evaluate the actual impact of the breach.
- 6 Describe the Australian Privacy Principles that apply to this situation.
- 7 Describe the security practices used by the Red Cross to comply with Australian privacy law.
- 8 What ruling did the Privacy Commissioner make regarding the breach?
- 9 Describe the improvements identified by the Australian Red Cross to ensure that donor information is protected in the future.
- 10 Reflect on a situation in which you have provided personal data using an online form. What expectations of privacy did you have in that situation?



4.1.5
Australian
Red Cross
Blood
Service
data breach

Exploring processes for secure data storage and exchange

A digital solution that exchanges data over the Internet needs to consider three main security risks, known as the 'CIA triad':

- *confidentiality* – the possibility that a person not authorised to see the data will observe the data in transit
- *integrity* – the possibility that someone may intercept the data message and modify it in a malicious way or replace it with a new message
- *availability* – the possibility that someone may interfere with the transmission to prevent data packets from reaching the intended destination.

The main techniques developers use to help protect against these threats are:

- *encryption* – scrambling the message to make it impossible to read in transit, but allowing it to be decrypted by the authorised recipient with a secret key
- *authentication* – verifying the identification of the sender by using a digital signature or authentication code
- *checksums* – ensuring data transmissions are transmitted accurately without accidental corruption or loss
- *hashing* – generating a secure message digest to verify data integrity of the complete message to protect against malicious interference.

Not all data needs to be secured during a digital data exchange, but personal information such as names, dates of birth, contact details and passwords is protected by privacy laws, and most security policies require that digital exchange of personal information is appropriately secured through encryption software. When using encryption software and code libraries to protect personal information, it is helpful to understand a little about the general approaches used by popular techniques.

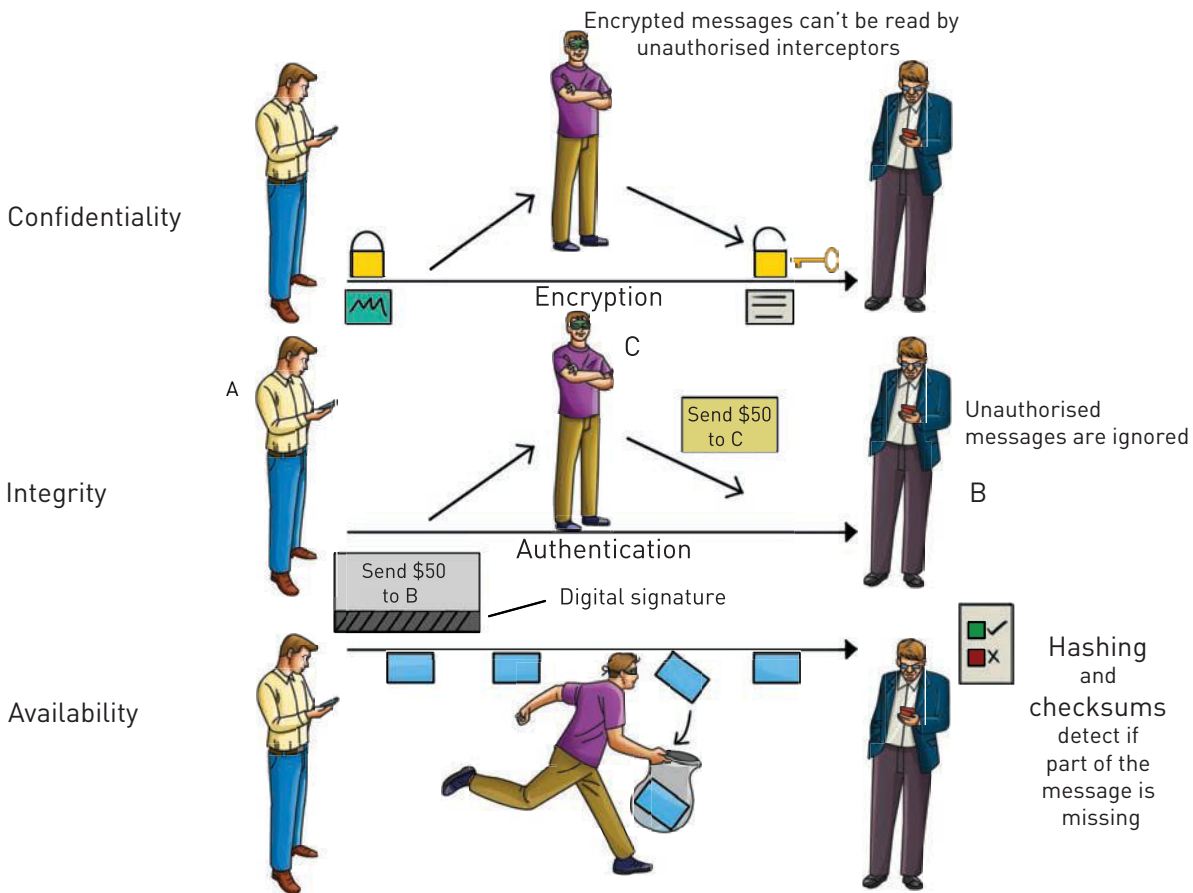


Figure 4.1.9 The CIA triad

Encryption and decryption

Encryption transforms data into an unreadable form. The unencrypted data is referred to as plain text, and encrypted data is referred to as ciphertext. The process to encrypt the data usually involves a repeatable process or algorithm and an encryption key. Anyone who knows the encryption algorithm and the encryption key can encrypt and decrypt messages and exchange data securely.

There are many different encryption algorithms. Well-known and popular algorithms include:

- TripleDES – used in automated teller machines (ATMs) to encrypt PINs
- Blowfish and Twofish – efficient open-source encryption algorithms used for application-level encryption
- AES – Advanced Encryption Standard
- PGP – Pretty Good Privacy, often used for encrypting emails
- RSA – Rivest-Shamir-Adleman public key algorithm, developed by MIT and used for encrypting most data sent over the Internet.

These encryption methods are designed for a complete package of data. Sometimes encryption is also required for data streams, which can be secured with alternative encryption methods such as RC4 (used for wireless transmissions) and Snow (used for mobile transmissions).

For detailed explanations of an extensive range of encryption algorithms, visit the asecuritysite.com website, created by cybersecurity expert Bill Buchanan.

Some password vaults are protected using a method called Honey Encryption. This is not an encryption method, but a strategy to confuse hackers. If it detects a hacking attempt, the Honey Encryption software will provide the hacker with fake data to make them think the hacking attempt was successful. This slows down the hacking process considerably.



TripleDES

TripleDES is a more secure version of the US Government DES security protocol, and encrypts data three times using a different key for at least one of the encryption iterations. Although popular for securing financial transactions, it is slower than most other methods as the encryption process is repeated three times.

Blowfish and Twofish

Blowfish and Twofish are unpatented open-source encryption algorithms. Blowfish splits a message into blocks of 64 bits and encrypts each block individually. It is one of the faster encryption algorithms and is often used in e-commerce payment gateways and password management systems. Twofish is a very similar algorithm that uses 128-bit blocks.

AES

The Advanced Encryption Standard (AES) uses a block cipher. It replaced DES as the US Government encryption standard in 2000 and is one of the most popular and secure encryption algorithms.

OpenPGP

Pretty Good Privacy (PGP) uses a combination of symmetric and public key encryption. In symmetric encryption, the same encryption key is used to encrypt and decrypt a message, and must be kept secure. In public key encryption, different keys are used for encrypting and decrypting a message. The private decryption key must be kept secure and is never transmitted, but the 'public' encryption key can be sent over the Internet. PGP uses public key encryption at the beginning of a session to exchange keys and encrypt the symmetric key. Then any messages that have to be sent during the session are encrypted using symmetric encryption. Because the session key has been encrypted using public key encryption, it can be sent securely with each message.

THE OPENPGP PROCESS

- 1 The sender creates a message and a random number to use as the encryption key.
- 2 The sender encrypts the message with the encryption key.
- 3 The sender requests a public key from the intended recipient and uses it to encrypt the encryption key so it can be sent with the encrypted message.
- 4 The recipient receives the encrypted key and message and extracts the encrypted encryption key from the start of the encrypted message.
- 5 The recipient uses their private key to decrypt the encryption key.
- 6 The recipient uses the decrypted encryption key to decrypt the message.

RSA

In the Rivest-Shamir-Adleman (RSA) algorithm, a public key is used to encrypt the message and a private key is used to decrypt it. The strength of the algorithm is attributed to the large prime numbers and keys used in the encryption calculations. However, the size of the integers also results in a slower encryption and decryption process.

The RSA encryption process is a complicated mathematical algorithm involving prime numbers and the Euclidean algorithm. If you have ever visited a website whose URL begins with 'https://', it is likely that an RSA algorithm has been used to encrypt the transmissions between your device and the secure web server.

Due to the increasing importance of cybersecurity, the University of Hawaii developed an RSA encryption activity for high school students as part of a STEM initiative. It helps to explain the mathematics behind RSA encryption.



Authentication

Authentication is the process of verifying that someone is who they say they are. A signature witnessed by a third party is a common method of authenticating that someone is eligible to take a driving test. Providing login credentials such as a username and password is a common way of authenticating a student on a school network, or a member of a gaming community. Often, the term ‘one step’ or ‘two step’ is used to identify how many levels of authentication are required at the application level before access is granted to the application.

When exchanging data over a network, it is sometimes important to authenticate an IoT device, a software API or a network server to ensure that they are not being impersonated in a malicious attack. A payment gateway, in particular, needs to authenticate that payment requests originate from a legitimate source. Digital signatures and application tokens are common strategies for authentication of users and devices over the Internet at the network level.

Digital signatures

A digital signature is data encrypted with the private key of a user using a mathematical scheme. It is a secure method of both encrypting and authenticating a data exchange. Early digital messages were authenticated using binary digital signatures, but binary representations differ for different operating systems. Modern digital signatures need to be supported across a range of different platforms, so an open standard such as Digest Access Authentication or XML Digital Signature is usually used for authentication of text, images and videos exchanged over the Internet.

Digest Access Authentication

Digest Access Authentication is part of the HTTP protocol standard and provides a way for websites to refuse requests from users that do not provide an appropriate response to a request for authentication. A user that attempts to bypass the login screen of a secure website will likely receive a ‘403 Forbidden’ error message if the website is protected by Digest Access Authentication.

After receiving an access request, a server creates a nonce value that can be used for one-time access to the server. It is usually based on the current date and time, combined with specific data about the server. The server sends the nonce value to the user application, which responds with a hexadecimal MD5 digest based on a valid username/password, URI, HTTP method and the nonce value. The MD5 digest is the digital signature for the request. The server compares the digest received with a calculated digest to determine whether or not to allow the access.

XML Digital Signature

An XML Digital Signature is used to authenticate all or part of an online data exchange, such as a web transaction or email exchange. XML tags are added to the original data source, including a reference to the part of the data to be authenticated, a link to the hashing and encryption algorithm(s) and the resulting hash value expected for an authentic copy. The information in the XML signature can be used to verify

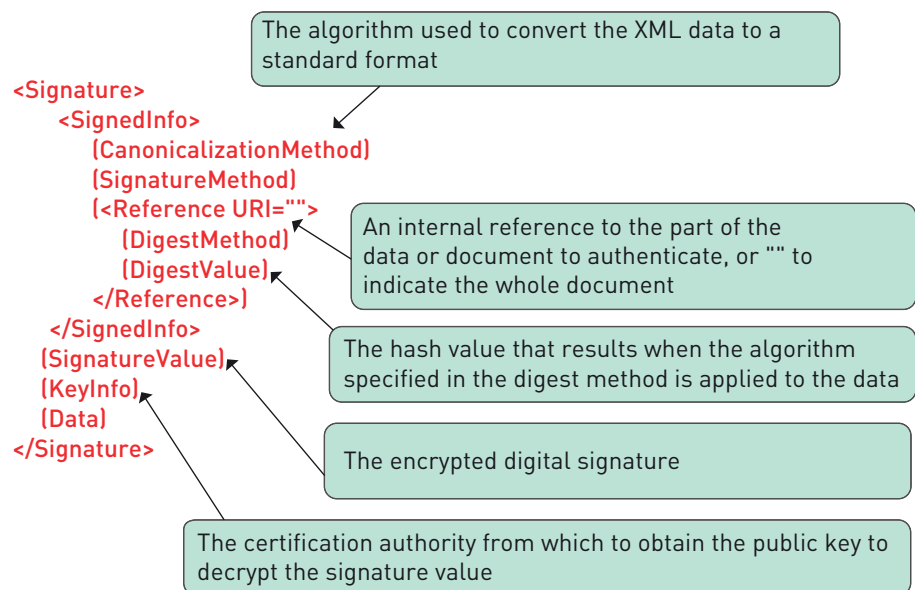


Figure 4.1.10 Components of an XML signature

that there have been no alterations to the original data within the XML document and that the source of the data is authentic.

Application tokens

An application token is a digital key that can be used to access protected data. Any program or user with the token can access the data. The token is associated with a specific user or device, and is generated once and used as often as required. An application token is usually represented as alphanumeric characters or hexadecimal codes, and is stored on both the client and the server components of a data exchange application. The single sign-on capabilities of many organisations are implemented using application tokens and allow users to log in once to gain access to multiple systems.

OAuth2

Open Authentication (OAuth) is an open standard for authorisation of applications and devices. OAuth2 is the latest version of the standard and is more secure. It uses access tokens rather than credentials to verify access requests, and can be combined with other security strategies such as SAML and OpenID. OAuth2 is the method used by many applications that offer you the option to log in via your Facebook or Google account. It is also used to restrict access to APIs.

MAC

A message authentication code (MAC) is the output from a mathematical calculation on a digital message, combined with a secret key. Its purpose is to authenticate the origin of the message as well as the integrity of the message itself. HMAC is a special type of message authentication code generated by a hash function.

SAML and OpenID

Security Assertion Markup Language (SAML) is an XML-based authentication process that redirects access requests to a central authentication server. OpenID is a JSON-based authentication service that generates a digitally signed and encrypted ID token using an OpenID provider. OpenIDs can be used to access multiple websites with a single login.

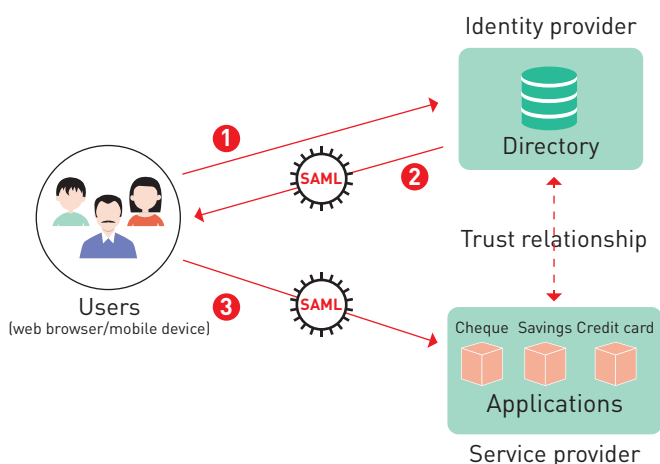


Figure 4.1.11 An SAML assertion is an example of an application token that can be used across multiple applications to authenticate a user.

Checksums

A checksum is like a fingerprint identification of a package of data. It can be applied to a small block of data within a data stream or to a complete file, and is often used to verify that data has not been modified during transmission or storage. A checksum algorithm inspects the binary representation of data and calculates a mathematical result based on the unique combination of binary digits. The binary representation of the result is referred to as the checksum. The checksum is then appended to the data stream or file, and can be used to verify the contents of the data packet or file by applying the same mathematical calculation and comparing the stored and calculated checksums.

Methods for generating checksums include:

- parity bits – used for detecting network errors in a small data block
- cyclic redundancy check – used for detecting data storage errors in a sequence of data blocks
- file checksum – used for checking that a data transfer is completed successfully.

Antivirus software often uses a file checksum to detect whether files have been modified by malicious software, and backup verification processes use checksums to determine if a system backup has been successfully copied to backup media.

Hashing

Hashing algorithms create a fixed-length hexadecimal value, called a digest or hash, for a variable-length set of data, based on mathematical calculations on the binary values used to represent the data. A small change in the data results in a completely different hash value, so the hashing algorithm can be used to authenticate a digital signature or verify the integrity of a large block of data. Hashing can also be used as a secure method of storing passwords. Instead of storing a password in plain text, the digest is stored instead. The more complex the hashing algorithm, the more likely it is that it will result in a unique hash value and the harder it is to crack. However, complex hashing algorithms require a lot of processing power and can slow down applications.

A cryptographic hash is similar to a checksum except that it is designed to thwart malicious alteration, rather than accidental transmission errors. The success of a hash algorithm can be appraised by evaluating the following characteristics:

1 Reliability

When applied to a specific message, the same hash digest should be generated every time. Otherwise it won't be a reliable integrity check.

2 One-way (resistance to pre-image attacks)

It should be impossible to reconstruct the message if only the digest is known. Otherwise the algorithm won't protect the confidentiality of the message.

3 Collision resistance

It should be nearly impossible to generate two identical digests from two different messages, no matter how similar or dissimilar. Otherwise it would be possible to forge an authentication code or slightly modify the message.

4 Speed

Fast computations minimise the performance overheads of security processes. Sometimes security requirements need to be balanced with processing capability.

Depending on the application, some of these criteria are more important than others. It is important to select a hash function that is strong in areas that are critical to a specific application.

Two common hashing algorithms are MD5 and SHA-1 but, as processing capabilities of computers advance, more effective hashing algorithms are developed. SHA-256 is now considered one of the most secure hashing algorithms but MD5 is sometimes used when speed is important.

MD5

Message-Digest-5 (MD5) is a one-way hash function used to create digital signatures, encrypt passwords and verify data integrity. It has been in widespread use since 1991 but, due to advances in computer processing power, it is no longer collision-resistant against a brute-force attack. Many developers still use MD5 for password encryption as it is reasonably fast and meets some of the requirements for a secure hash function. Although MD5 is no longer recommended for authentication, many security experts claim it is still suitable for some purposes, such as password verification, because it is resistant to pre-image attacks.

SHA

SHA-1 is based on, and replaced, MD5. It has been used extensively since 1995 to validate credit card transactions and software downloads, and is the default security setting for many Internet applications. For a long time, SHA-1 was considered the most secure hashing algorithm, and it has been used extensively by Microsoft for a range of purposes, including software protection on Xbox consoles. However, in 2017, security researchers proved that the processing capabilities of supercomputers could exploit a weakness in the SHA-1 algorithm to generate the same checksum for two different PDF files. This means that, although it is still in widespread use, SHA-1 is no longer secure. SHA-256 is a more complex and secure version of SHA-1 and is recommended as the new replacement for SHA-1. For high-security applications, the slower but harder-to-crack SHA-512 hash function is recommended.



In the novel *Gulliver's Travels* by Jonathan Swift, a conflict between two nations is attributed to a difference of opinion about whether to break an egg at the little end first or the big end first. Those who believed the most convenient end to start with was the big end were referred to as Big-Endians, and they rebelled against the laws supporting the Little-Endian view.

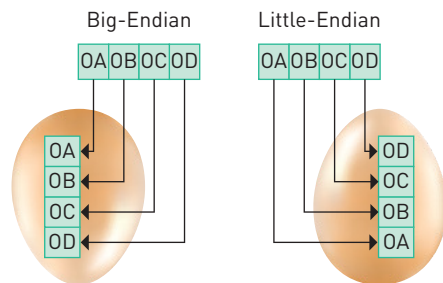


Figure 4.1.12 Big-Endian and Little-Endian data transfer

A similarly pointless dispute exists between computer architects, who cannot agree whether to store data bytes in memory in left-to-right or right-to-left order. This becomes a problem when transferring data over a network between computers with different architectures, as it is not always possible to know whether the sender is a Little-Endian or a Big-Endian. It is very difficult to reassemble data correctly without knowing the byte order so, when two systems exchange data at the byte level, they need to follow the Endian protocol defined by the network protocol. Internet standards specify Big-Endian, but not all network protocols follow this standard. If the Endianness of the processor is different to the network Endianness, then the byte order needs to be rearranged internally before data is sent over the network.

Some encryption and hashing algorithms, such as MD5 and AES, also need bytes to be in a specific order, so additional processing is required to swap bytes around if the Endianness of the operating system is different to the Endianness of the algorithm. In most situations, the difference in speed is measured in milliseconds, and this layer of complexity is hidden from the developer so Endianness can usually be ignored. Sometimes, however, when using encryption and hashing libraries, an Endianness parameter needs to be specified. It is helpful to know the Endianness of your hardware and try to match this to your encryption algorithm if possible. Most Windows and Linux systems are Little-Endian and most Apple and Unix systems are Big-Endian.

Investigating Internet transmission principles

When we use a digital solution that incorporates data from an online source, the data is exchanged between the local device and an Internet server. The process for transmitting this data is very complex and involves many different layers of interconnected hardware and software. The protocols to exchange data reliably across diverse networks and platforms have been developed and refined by independent organisations over many years. Some international standards organisations, such as the World Wide Web Consortium (W3C) and the International Organization for Standardization (ISO), continue to develop and refine standards for all levels of network communications.

- **W3C**

The W3C was founded by Tim Berners-Lee, who is credited with inventing the Internet. It is a well-respected international standards organisation responsible for obtaining worldwide consensus on protocols such as HTTP, FTP, HTML, CSS, XML, CORS, WCAG and ARIA, and is currently working on standards related to IoT applications, web payments and APIs for exchanging automotive vehicle information.

- **ISO**

Several years before the invention of the Internet, the ISO developed a seven-layer model for Open Systems Interconnection (OSI) to help conceptualise the different levels of network interaction that occur when applications transmit data. The OSI model provides a useful structure to define the responsibilities of hardware, operating systems and applications when transmitting data. The layers in the model are theoretical and are meant to be only a rough guide. The practical implementation of hardware and software will often combine several layers of the OSI model in a single function, particularly at the higher software levels.

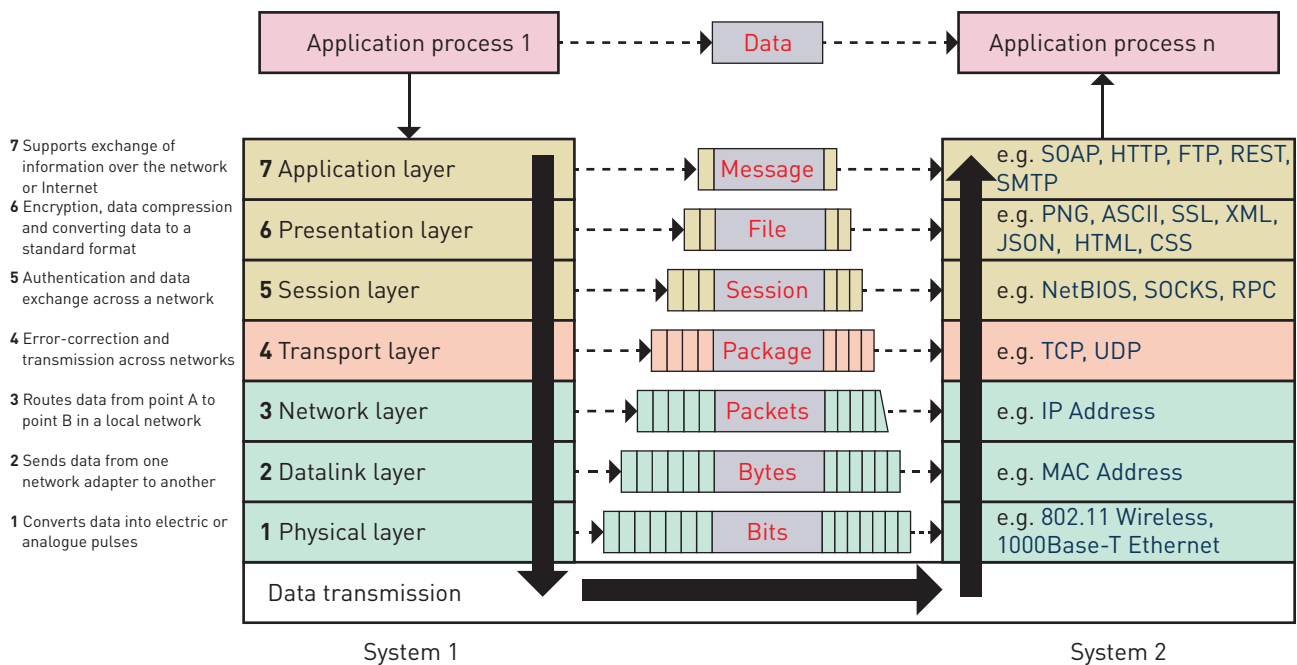


Figure 4.1.13 OSI model

Open-source groups such as the Internet Engineering Task Force and the Mozilla Foundation focus on standards for open and accessible Internet connectivity.

- **IETF**

The Internet Engineering Task Force is the standards body for the International Internet Society and is made up of respected IT professionals who identify and develop standards in response to needs identified by the society's members. It has published voluntary standards for FTP, email addressing, TCP/IP and JSON.

- **Mozilla Foundation**

The Mozilla Foundation was established to provide advisory services to support open web development. Development of the free Firefox browser was coordinated by Mozilla as an open-source project and the foundation continues to actively promote open-source development and provide input into development of open Internet standards.

Transmitting data

When data is transmitted from one device to another it needs to successfully complete a series of steps as it travels through the layers of the OSI model.

- 1 The data needs to be packaged and addressed before it can be transmitted.
- 2 It needs to be serialised into bits and each bit is sent one at a time.
- 3 Each bit travels through a physical cable or airwaves to the nearest connected device.
- 4 The nearest connected device collects the bits and checks each byte arrived safely, using a checksum calculation, before sending it bit by bit to the next device on the path.
- 5 At particular points in the transmission, a router needs to inspect the data packet to determine the next nearest network device to transmit the packet to.
- 6 If a packet takes too long to arrive, it will become stale and won't be transmitted.
- 7 This continues until the data packets begin to arrive at the intended location.
- 8 As each packet arrives, it is checked for corruption using a checksum.
- 9 If a packet is corrupted or lost, a replacement packet is requested.
- 10 Packets need to be buffered in memory until all packets arrive.
- 11 When all packets have been reliably received, they are reassembled.
- 12 If required by security protocols, additional calculations on the data may be required to verify its integrity and authenticity.

- 13 The data package needs to be converted into the appropriate format for the destination device and, if it has been encrypted for secure transmission, it needs to be decrypted.
- 14 Then the data can be processed by the application.

These are the basic network processes that attempt to ensure that data is sent reliably between two networked devices. Protocols and network processes are managed by the devices and networks that the data travels through. In most situations, security steps are required to authenticate and decrypt the data. Most steps only take fractions of a millisecond to complete, but for the user or application waiting for the data to arrive it may seem to take forever. Understanding the process of data transmission and network principles related to data transfer helps developers to test and propose performance improvements for digital data exchange solutions.

Data compression is a technique used to encode data efficiently so as to reduce the number of bits required to represent data. This has the desirable effect of reducing the amount of data storage required and the time to transmit the data over a network connection. There are a number of specialist data-compression algorithms suitable for different types of data such as text, images, audio and video. Some of these algorithms focus on reducing the size of a file while retaining data quality (lossless), and others focus on reducing the size of a file while accepting some loss of data quality (lossy). In digital data exchanges of text data, lossless compression such as Huffman encoding is usually preferred, but lossy compression such as H.264, MP3 and JPEG is often applied to large video, audio and image files to minimise file sizes before uploading to a website.

The standard compression method for data transmissions recommended by the Internet Engineering Task Force (IETF) is DEFLATE. It is a lossless compression method that combines the LZ77 and Huffman compression algorithms and uses a Hash Table to speed up the compression/decompression process. To compress the data, duplicate strings in the data stream are replaced with a shorter, fixed-length hash value. To reconstruct the data, the hash value is used as a lookup to a table containing the original data sequence.



4.1.7 How to use the Huffman encoding algorithm

Some important network principles to understand include latency, jitter, QOS guarantee and timeliness of delivery.

Latency

Latency is the delay between the time when an application requests data until it receives the data. Latency can be influenced by a number of factors:

- **Serialisation**

The time taken to physically move data onto the network will be determined by the size of the packet and the speed of the network interface.

- **Propagation**

The Internet is a **heterogenous** collection of high-speed and low-speed connections, some more reliable than others. The type of network media and the distance between the device requesting the data and the location of the server or sensor where the data is stored will influence the time it takes for data to travel along the network connection.

- **Switching**

As the data packets move through the network, they will need to be inspected and processed by switches and routers at various points along the network path. Each time a packet is inspected, additional delays are incurred. There is a mix of old and new, slow and fast equipment throughout the Internet. Switching latency for Internet communications can vary from one transmission to another depending on the network pathway the data travels along.

- **Queuing**

During busy times, network devices may need to buffer packets before they are inspected, adding additional delays each time this occurs. If there is a lot of traffic on a particular network link, data

collisions can result in lost packets. Data collisions occur when the bits from two data packets overlap and both packets are discarded. This requires both packets to be re-sent and results in additional delays.

To help manage network latency, digital data exchange developers often use an asynchronous programming technique such as Asynchronous Java and XML (AJAX).

Jitter

Jitter, or packet delay variance, is the irregular latency in data exchanges. In order for two devices to send data over a network connection, they need to agree on the rate at which the data is sent and read. This occurs on the data link layer of the OSI model. At a basic level, such as a simple serial connection used by IoT devices, the data rate is the **baud rate** established for the transmission. The serial receiver expects to receive data at this specific rate or bit errors will occur. At the data link layer, jitter can be caused by electronic noise and clock timing errors.

In higher bandwidth Internet connections, data compression, multiplexing and other advanced network communication techniques are used to increase the bit rate. When exchanging data over an Internet connection, jitter is measured at a higher level on the OSI model and impacts the exchange of data at the transport layer. Jitter at the transport layer is usually caused by network congestion as packets from the same data stream can be sent via different routes.

Some digital solutions can tolerate jitter, but in **synchronous** data exchanges such as video conferencing, online gaming and voice over IP (VoIP), fluctuating jitter can have a big impact on the user experience. During an online game, jitter may cause the player avatar to pause and jump around or skip over some scenes or dialogue, and in a video it can cause pixelation or poor audio quality. To

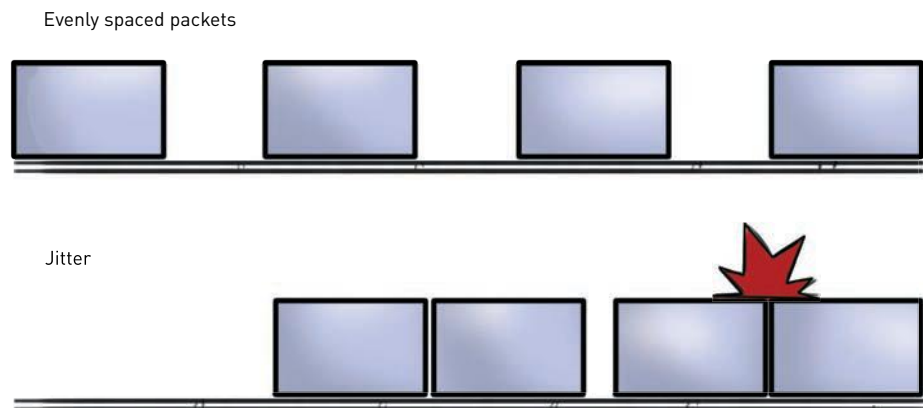


Figure 4.1.14 Jitter is a variation in the delay of received packets.

combat jitter, developers use buffering techniques. The small delay that occurs when you watch a video on YouTube is to buffer the video data so that it will play smoothly. Buffering uses memory to store packets as they arrive and deliver them to the application at more regular intervals. Some smaller devices such as IoT microprocessors have limited memory and therefore limited buffering capabilities. Microprocessor boards that support a Universal Asynchronous Receiver Transmitter (UART) such as the Raspberry Pi will have a small memory area called the serial buffer for buffering incoming data from serial ports.

Quality of Service (QoS) guarantee

Quality of service network standards provide a mechanism to prioritise some packets over others to reduce latency and/or packet delay variance.

The three types of QoS guarantees are:

- best-effort service – no guarantees
- differentiated service – some traffic is treated better than other traffic
- guaranteed service – some network bandwidth is reserved for use only by mission-critical applications.

These service levels are required because applications exchange data in different ways. Video delivery services such as Netflix need to transfer a large amount of data in one direction. Voice over IP telephony systems need to transfer a small amount of data in two directions with minimal delay. In these applications, speed is of high importance and some packet loss can be tolerated. Email applications can tolerate delays but not packet loss. If both systems share the same network, QoS guarantees provide a way for serialisation of small telephony packets to take priority over serialisation of large video packets, and email can be delivered reliably using a best-effort service.

QoS protocols also enable private network administrators to manage network traffic more effectively by allowing mission-critical applications to receive a higher network priority and greater bandwidth than less critical applications. As applications have moved from private networks, owned by organisations, to the World Wide Web, shared by an international community, Internet service providers have used QoS protocols to establish differentiated Internet services. In most cases, users can choose to pay a higher subscription fee to access a guaranteed bandwidth that reduces delay and packet variance for specific types of applications. Availability of these distributed Internet services has fuelled the so-called net neutrality debate, over the question of whether all packets should be treated equally. The need for differentiated services needs to be balanced with social and ethical considerations to prevent an 'Internet divide' between those who can afford to pay for higher bandwidth and those who cannot.

The rollout of the National Broadband Network (NBN) replaces Australia's best-effort Internet service with a distributed-services QoS model that treats voice over IP and pay TV services more favourably than other traffic types. In an ethernet data packet, the priority is identified with a 3-bit Priority Code Point (PCP) field at the start of the packet. The classes of traffic on the NBN are:

- TC_1 – voice services with strict delay requirements (PCP 5)
- TC_2 – interactive video services (PCP 4)
- TC_3 – guaranteed bandwidth for enterprise data services (PCP 3 or 2)
- TC_4 – best effort (PCP 1).

NBN devices assign each class of traffic a different queue to ensure that premium services, delay-sensitive traffic and rate-sensitive traffic can be delivered at high performance speeds while still supporting normal Internet traffic such as browsing and email using a best-effort approach.

Net neutrality is the principle of an open Internet that allows Internet users the freedom to:

- access content
- use applications
- attach personal devices
- obtain service plan information

without fear of censorship, blocking or anti-competitive interference by ISPs. Net neutrality laws were first implemented in the US to combat discriminatory management of broadband Internet traffic by ISPs that were attempting to block or slow down traffic from competing services. Some lobbyists consider that premium Internet services that deliver faster Internet speeds to customers who can afford it are also a threat to net neutrality.

In many countries, net neutrality laws or the actions of regulatory authorities prevent or discourage ISPs from interfering with content transmitted via their services to ensure that content or applications are not blocked or delayed arbitrarily. In 2017, the Trump administration's decision to repeal US net neutrality laws and replace them with transparency regulations enforced by the Federal Trade Commission made headline news throughout the world. Australia has no net neutrality laws but the Australian Communications and Media Authority (ACMA) is responsible for ensuring that ISPs comply with consumer law discouraging discriminatory behaviour.



Timeliness of delivery

Timeliness is the currency of a data packet at the time it is received. Timeliness is usually measured by comparing the timestamp of the packet to its delivery deadline and counting the number of packets that

arrive after the deadline. Many applications are not time-sensitive, but for real-time applications any delay in milliseconds may require adjustments in the way the data is processed. For example, in safety-critical transport systems, a delay in transmitting data from a control sensor to stop a vehicle may require additional braking pressure to stop the vehicle in time. In a real-time stock exchange system, data that arrives after its deadline is discarded because it is no longer accurate. Timeliness is a critical network principle for real-time or near real-time applications.

Data packets with timeliness constraints have hard deadlines if the data is discarded when late, and soft deadlines if the data can be used but is not as accurate if received late. Data with no deadlines is usually delivered with a best-effort approach.

Internet protocols

The Internet is an interconnected network of cables, routers and switches that transmit packets of data. The internal format of each data packet travelling through the Internet is largely dependent on the application-specific protocols agreed between the sender and receiver, but the external wrapper or envelope that the data travels in follows protocols agreed by international standards for transfer of data across the Internet. The main protocol used on the Internet is HTTP, or Hypertext Transfer Protocol.

HTTP

Hypertext Transfer Protocol (HTTP) is a stateless request/response protocol, which means that each message received is treated as an independent exchange between a client and a server and is not reliant on a successful login or menu selection from another web page. Each request message can be understood in isolation.

The HTTP client can be a web page, IoT device, traffic camera, mobile app or desktop app that sends an HTTP request. An HTTP server is the web server, robot, network component or interactive media platform that responds to the HTTP request. Each HTTP request contains a method type and a Uniform Resource Identifier (URI). The HTTP response includes a success or error code and a series of response messages containing the requested data. For example, in response to an HTTP request containing a GET method and a web page address as the URI, a web server HTTP response would typically contain a success code, followed by the web page data. If there is no data available at the location specified, a '404 Not Found' error would be sent instead.

Table 4.1.1 HTTP methods

HTTP METHOD	DESCRIPTION	POSSIBLE RETURN CODES
GET	Retrieve metadata and data from a location specified by the URI e.g. to obtain a list of students at www.myschool.com.au GET http://www.myschool.com.au/students HTTP/1.1	200 Success 404 Not found
HEAD	Retrieve metadata only from the location specified by the URI	200 Success 404 Not found
POST	Submit data to a message board, form-handling process or database e.g. to upload a timetable POST http://www.myschool.com.au/timetable HTTP/1.1	201 Created 204 No content 404 Not found 409 Conflict
PUT	Add data at the location specified by the URI e.g. to add a new student record PUT http://www.myschool.com.au/student/1234 HTTP/1.1 <student><fname>Josh</fname></student>	201 Created 204 No content 404 Not found

HTTP METHOD	DESCRIPTION	POSSIBLE RETURN CODES
DELETE	Send a request to the server to delete the data at the location specified by the URI. e.g. to delete a student record DELETE http://www.myschool.com.au/student/1234 HTTP/1.1	200 OK 202 Accepted 204 No content
TRACE	List proxies or gateways encountered on a request path or keep a record of actions taken for diagnostic purposes	200 OK
CONNECT	Initiate a connection with a secure SSL tunnel	200 Success
PATCH	Update or modify data at the location specified by the URI. The modified data is usually provided in a JSON Patch or XML Patch format. e.g. to update a student record UPDATE http://www.myschool.com.au/student/1234 HTTP/1.1 <student><fname><from>Josh</from><to>Joshua</to></fname></student>	200 OK 404 Not found
OPTIONS	Return the list of HTTP methods supported by the server.	200 OK

HTTPS

The ‘S’ in HTTPS stands for ‘secure’. HTTPS is a secure version of HTTP that uses Transport Layer Security (TLS) or Secure Socket Layer (SSL) protocols to encrypt the transmission using public key encryption methods. TLS is often referred to as SSL/TLS as it is essentially just a new version of SSL with some extra features. When a web page or data is requested via HTTPS, the web server will first send an SSL Certificate to the browser. This contains the public encryption key to begin the data exchange. The public key is then used in an ‘SSL handshake’ to establish the secure session. The SSL handshake process is:

- 1 The client node sends the SSL version number, a list of possible cipher suites that could be used and a list of possible compression settings to the server.
- 2 The server node selects the most secure settings supported by both client and server, and replies with the selected SSL version number, cipher and compression settings.
- 3 The client node authenticates the SSL certificate and sends an encrypted message back to the server using the public key from the SSL certificate. The message contains a secret that can be used for authentication during the session.
- 4 The server node uses its private key to decrypt the secret message.
- 5 Both client node and server node confirm that the session will be encrypted and begin exchanging encrypted data.

FTP

File Transfer Protocol (FTP) is the protocol used to exchange files over the Internet. It is used by web developers to upload files to a web server. It can also be used to download files from an online server to a client machine. Due to security vulnerabilities, FTP is blocked on most web servers.

FTP connections are initiated with a username and password authorised to connect to the server with the FTP protocol. Once the FTP session is established, files can be exchanged freely between the client and the server. The FTP session is not encrypted. The password and all files are sent as plain text, so FTP is considered very insecure. If an encrypted connection is required, SFTP can be used instead of FTP so that the FTP password is not sent as plain text. Some servers offer an anonymous FTP service that allows unauthenticated access to the server.

The addresses of FTP sites usually begin with ‘ftp://’ instead of ‘http://’, and ‘ftp’ instead of ‘www’ – for example, ftp://username@ftp.example.com/. File transfers can be requested at the command line using standard FTP commands such as ‘open’, ‘get’, ‘put’ and ‘close’. DOS-like commands such as ‘cd’, ‘dir’, ‘mkdir’ and ‘rmdir’ can be used to navigate, create and delete folders on the FTP server. Graphical FTP software, or FTP plug-ins to web development software such as Adobe Dreamweaver, allow files to be dragged between client and server windows once an FTP session has been established.

ACTIVITY 4.1.3

Some software sites provide anonymous FTP access to their software. Visit the Adobe FTP download site to explore how FTP works by downloading a test file, and then click on Folders to explore the folder structure of the Adobe download site.

Explore how FTP works using a command line tool:

- 1 Open the command line interface for your operating system – in Windows, for example, open a command window, or in Mac OS open a terminal session.
- 2 Navigate to a folder where you have permission to save files. The files you download during your FTP session will be saved here.
- 3 Type 'ftp' at the command line to open the FTP utility:

```
ftp>
```

- 4 Open a connection to the FTP site with the FTP open command:

```
ftp> open ftp.adobe.com
```

- 5 Enter the anonymous login details:

```
user: anonymous; password: ftpuser@adobe.com
```

```
User (ftpgen.wip4.adobe.com: (none)) : anonymous
```

```
331 Please specify the password
```

```
Password: ftpuser@adobe.com
```

```
230 Login successful
```

- 6 Use 'dir' and 'cd' commands to navigate the FTP site:

```
ftp> cd pub
```

```
250 Directory successfully changed
```

```
ftp> dir
```

- 7 Use the get command to download a file:

```
ftp> get Stage
```

```
200 PORT command successful. Consider using PASV.
```

```
150 Opening BINARY mode data connection for Stage (24 bytes)
```

```
226 Transfer complete
```

```
ftp: 24 bytes received in 0.00Seconds 24.00Kbytes/sec
```

- 8 Close the FTP connection and exit the FTP utility:

```
ftp> close
```

```
ftp> quit
```



4.1.8
Adobe FTP

VPN

A virtual private network (VPN) is an encrypted connection between two network devices. All the traffic sent between the two devices is encrypted until the connection is closed. Establishing a VPN connection is commonly referred to as 'tunnelling', and can be implemented with a variety of different security protocols, such as SSL at the application layer, or IPSec at the network layer. Because most VPN security protocols use public key encryption, a VPN connection ensures both data integrity and authentication during the data exchange. VPNs are used for a range of purposes, including accessing sensitive data remotely and exchanging data securely between applications.

Methods for exchanging data

Data can be exchanged over the Internet using a variety of methods. At the network layer, the data stream can be unicast, broadcast or multicast. Unicast transmission occurs when the data exchange takes place between two specific devices (for example, an HTTP web page or SMTP email). Broadcasting occurs when a server transmits a datagram (such as a game service UDP) to the broadcast address of a network so that it is received by all devices on the network. Multicasting allows a server (such as a video server) to exchange data with multiple specific client nodes at once.

At the session or application layer, the data exchange can be synchronous or asynchronous, and at the presentation layer the data can be published in a variety of formats. Synchronous exchanges use a conversational approach whereby the sender and receiver take turns sending and acknowledging data exchanges. Asynchronous exchanges such as AJAX do not wait for a response. The format of the exchanged data will also vary depending on the application. Two widespread data formats are JSON and XML.

RESTful services

Representational State Transfer (REST) is a client-server architecture for manipulating an Internet resource at a specific URI. It will generally use JSON or XML formatted data, but it is also possible to use CSV or other open data formats with a RESTful service. REST documentation usually uses the HTTP verbs GET, PUT, POST and DELETE, as these are standard terms easily understood by developers. Like HTTP, it also assumes a stateless approach, so that each message is self-descriptive and contains all the information required to execute a request independently of any messages sent before or after the current message. Another characteristic of a RESTful service is that it assumes that data will be cached, and that any changes made could be applied to an intermediate copy of the data resource. This makes the solution scalable and provides the option for the client to make decisions about the data without consulting the server. However, this also means that any requests that modify the data stored in the cache are considered unsafe, and **idempotent** methods that preserve the state of the cached resource are preferred as they are considered safe.

The RESTful approach is suitable for interacting with open data sources where changes to the source data are not required, but it is less suitable for applications that need to update data or keep track of the application state, such as financial transactions and user login status. Some organisations still use a REST architecture for updating web-based resources where the application resides within a private organisational network.

AJAX

Asynchronous JavaScript and XML (AJAX) is an asynchronous web development architecture used to build interactive web applications. The main idea is that it allows small sections of the web page to be exchanged in the background without the need to reload the page constantly, and facilitates the development of a responsive website application capable of responding quickly to user interactions. Even though the X in AJAX stands for XML, AJAX can be used with data in many different formats. Google Maps is a good example of a web application built using an AJAX architecture.

For example, one script requests the Google Map:

```
<script src="https://maps.googleapis.com/maps/api/js?key=YOUR_KEY&callback=myMap">
</script>
```

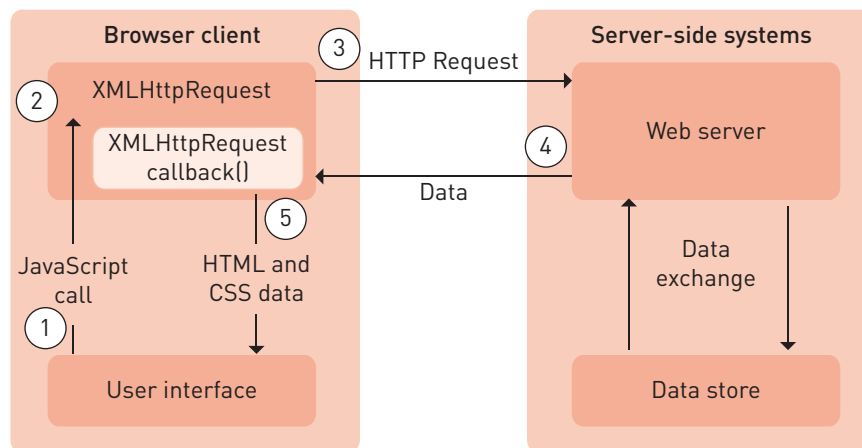


Figure 4.1.15 AJAX



4.1.9 Best practices for building RESTful web services

Another script processes the data when it is returned:

```
<script>
function myMap() {
  var mapProp= {center:new google.maps.LatLng(-22,145), zoom:5};
  var map=new google.maps.Map(document.getElementById("googleMap"), mapProp);
}
</script>
```

In a synchronous data exchange, the application sends a request for data and waits for the data to be sent. In an asynchronous data exchange, the application uses a request callback method. This means that the application can present part of the information and respond to other user requests while waiting for the data to arrive. AJAX can be used to update parts of a web page with either JSON or XML data when it arrives from a callback request.

JSON

JavaScript Object Notation (JSON) is a data exchange protocol based on JavaScript. Although it was originally created for JavaScript, it has become a de facto standard for exchanging key-value data pairs; JSON libraries now exist for a range of programming languages. The JSON protocol allows a program to establish a connection to a live website and interact with online data. The JSON API on the client side manages the data exchange using HTTP GET and POST requests. The data API on the web server responds to the data requests to send the data as a JSON object. Some data APIs allow parameters to be used to control access to the dataset or query a subset of the data. The parameters are usually passed to the web server as key-value pairs embedded in the URL request.

JSON data follows this format:

```
{ "key1":"value1", "key2":"value2", "key3":"value3", ... }
```

Each key is a different category of data. For example, species of pygmy possums in the wildlife open dataset managed by the Queensland Department of Environment and Heritage are represented in JSON like this:

```
{ "Species": [
  { "TaxonID": 917,
    "ScientificName": "Cercartetus caudatus",
    "AcceptedCommonName": "long-tailed pygmy-possum",
    "FamilyName": "Burramyidae",
    "FamilyCommonName": "pygmy-possums" },
  { "TaxonID": 918,
    "ScientificName": "Cercartetus nanus",
    "AcceptedCommonName": "eastern pygmy-possum",
    "FamilyName": "Burramyidae",
    "FamilyCommonName": "pygmy-possums" },
  { "TaxonID": 919,
    "ScientificName": "Cercartetus sp.",
    "FamilyName": "Burramyidae",
    "FamilyCommonName": "pygmy-possums" }
] }
```

Source: <https://environment.ehp.qld.gov.au> © State of Queensland CC BY 4.0

JSON data can be accessed within program code as a JavaScript object using JSON libraries, or converted to an equivalent data structure suitable for the specific programming language used, such as an associative array, indexed list or dictionary tuples.



ADVANCED DATA STRUCTURES

Different programming languages have different types of data structures and manage memory storage in different ways. Sometimes the same or a similar data structure has a different name depending on which programming language you are using. Some common data structures are explained in table 4.1.2.

Table 4.1.2 Advanced data structures

DATA STRUCTURE COMMON NAME	DESCRIPTION AND EXPLANATION	PROGRAMMING LANGUAGE-SPECIFIC NAMES
Array	A list of items that can be accessed using the position of the item in the list, which is referred to as the item index. In most programming languages, the index of the first item in the list is 0. The value of individual items in the list can be changed through assigning values, using the index as a pointer to the item that needs to be changed.	Array, list Note: some programming languages require all items in an array to be the same data type; other programming languages allow heterogenous data types.
Associative array	A list of items that can be accessed using a descriptive name, which is referred to as the associative index. The value of individual items in the list can be changed through assigning values, using the label as a pointer to the item that needs to be changed.	Associative array, dictionary, named tuple
Tuples	A list of values such as x,y coordinates, RGB colour values or latitude and longitude. The data values in the tuple cannot usually be changed individually and are treated like a complete set.	Tuple, record, list
Key value pairs	A pair of values in which one item is the key and acts as a lookup for the second value, which is the data.	Object, tuple

Advanced data structures are formed when one type of data structure is nested inside another. For example, an indexed list may be used to store map locations in a game where each item in the list is a grid location on the game map. If each map location needs to keep track of items available in that location, an indexed array may be used for the item list. This results in arrays of arrays, and two indices are required to identify a single item. Where nested lists are used, nested iteration is required to step through each item in the inner and outer lists.

XML

Extensible Markup Language (XML) is an extension to HTML that allows developers to create data tags for storing formatted data in a web page. It follows a similar hierarchical structure to the HTML Document Object Model (DOM). In an HTML document, the different parts of the web page are represented in a hierarchy. Div tags of a web page are contained within the body tag, which is contained within the HTML tag. XML data is also represented as a hierarchical structure with values contained within key tags, which are contained within record tags, which are contained within dataset tags.

XML data follows this format:

```
<dataset>
  <record>
    <key1>value1</key1>
    <key2>value2</key2>
    <key3>value3</key3>
  </record>
</dataset>
```

The first tag in the XML dataset is referred to the root element. It is the container for all subsequent elements. Each key tag represents a different category of data, and all the data about one item is grouped together using XML tags that define the record. The wildlife dataset explored on page 305 can be represented in XML like this:

```
<Species>
  <Species>
    <TaxonID>917</TaxonID>
    <ScientificName>Cercartetus caudatus</ScientificName>
    <AcceptedCommonName>long-tailed pygmy-possum </AcceptedCommonName>
    <FamilyName>Burramyidae</FamilyName>
    <FamilyCommonName>pygmy-possums</FamilyCommonName>
  </Species>
  <Species>
    <TaxonID>918</TaxonID>
    <ScientificName>Cercartetus nanus</ScientificName>
    <AcceptedCommonName>eastern pygmy-possum</AcceptedCommonName>
    <FamilyName>Burramyidae</FamilyName>
    <FamilyCommonName>pygmy-possums</FamilyCommonName>
  </Species>
  <Species>
    <TaxonID>919</TaxonID>
    <ScientificName>Cercartetus sp.</ScientificName>
    <FamilyName>Burramyidae</FamilyName>
    <FamilyCommonName>pygmy-possums</FamilyCommonName>
  </Species>
</Species>
```

Source: <https://environment.ehp.qld.gov.au> © State of Queensland CC BY 4.0

XML data can be accessed from within a program using a variety of different methods, including data binding, downloading XML data and reading an XML data stream. In desktop applications, data binding is often used to associate specific XML tags with equivalent programming data structures. An XML schema is generated to define the structure of the XML document, and the XML schema is then mapped to equivalent components of an internal data structure defined by the application. Most programming languages have XML libraries or classes that define the syntax required to create the code for the bindings and synchronise the XML data source with the application dataset.

For example, using System.Xml:

```
private void ReadXmlButton_Click(object sender, EventArgs e)
{
    string urlPath = "https://environment.ehp.qld.gov.au/
species/?op=getspecies&family=burramyidae&f=xml";
    PygmyDataSet.ReadXml(urlPath);

    dataGridView1.DataSource = PygmyDataSet;
    dataGridView1.DataMember = "Species";
}
```

If the XML data is reasonably static, it can be easier and more reliable to download a copy of the XML data to the local folder structure of the application and process the data as a local file.

The Python programming language has a special tree data structure in its XML library that can be used to load a small XML document into memory.

For example:

```
import xml
tree = xml.etree.ElementTree.parse('pygmy_data.xml')
```

Python also has an 'xmldict' library that allows a developer to load an XML file directly into a dictionary data structure.

For example:

```
import xmldict
with open('pygmy_data.xml') as fd:
    doc = xmldict.parse(fd.read())
```

VB.Net, C# and a number of other programming languages include three main XML methods:

- **xmlDocument**, which reads an entire XML data file into a hierarchical object in memory
- **xmlReader**, which reads the XML document line by line from a local file to conserve memory
- **xmlTextReader**, which navigates an XML data stream using a file cursor and allows it to be processed element by element.

TEST YOUR KNOWLEDGE

- 1 Give an example of a data exchange architecture and explain how it works.
- 2 What is the purpose of a network protocol? Explain the difference between HTTP and FTP.
- 3 What does 'API' stand for? Describe the main purpose of an API.
- 4 Explain why it might be helpful to use an ER-diagram to represent an existing dataset.
- 5 What was the purpose of the strengthening of Australian privacy laws in 2014?
- 6 List the 13 Australian Privacy Principles from Schedule 1 of the *Privacy Amendment (Enhancing Privacy Protection) Act 2012*, which amends the *Privacy Act 1988*.
- 7 Describe the three parts of the CIA triad and recommend security strategies that can be effective in each part.
- 8 Explain each of the following data exchange protection measures.
 - a encryption
 - b authentication
 - c checksums
 - d hashing
- 9 Identify one popular encryption/decryption method and give an example of how it is used in industry.
- 10 Identify some strategies that are used for authentication of text, images and video exchanged over the Internet.
- 11 Explain the major difference between a digital signature and application tokens when used for authentication.
- 12 Identify three methods of generating checksums.
- 13 Describe an example of the use of checksums in industry.
- 14 The success of a hash algorithm can be appraised by evaluating four characteristics. Identify these characteristics.
- 15 Explore an example of one hashtag algorithm that is used in industry.
- 16 Identify at least one protocol used in each of the seven layers of the OSI model.
- 17 What is the series of steps taken when data is successfully transmitted from one device to another as it travels through the layers of the OSI model?
- 18 Latency is the delay from when an application requests data until it receives the data. Describe and explain a number of factors that can influence latency.
- 19 What does 'QoS' stand for? What does QoS allow private network administrators to do more effectively?
- 20 What are the three types of QoS guarantees?
- 21 Identify the main protocol of the Internet and describe how it works to transport data.
- 22 What does 'HTTPS' stand for and how does it differ from HTTP?
- 23 REST is a method for exchanging data. Identify and explain one other method.



APPLY YOUR KNOWLEDGE

- 1 When receiving digital data, it is important for a recipient to verify two properties of the data:
 - integrity – that the message was not accidentally or maliciously modified during transmission
 - authenticity – that the message received by person B was sent by person A.

Evaluate the following security techniques to determine the extent to which they fulfil these two requirements.

- a digital signature
 - b hash
 - c MAC
- 2 Exchanging open data has the power to change the world.
 - a Review Australia's Public Data Policy. One of the commitments in this policy is that government entities will 'make non-sensitive data open by default to contribute to greater innovation and productivity improvements across all sectors of the Australian economy'. Another commitment in this policy is that governments will share data securely to become more efficient.

- b Create a mind map exploring the opportunities for innovation and productivity improvement that could occur as a result of the Public Data Policy. Consider opportunities for innovation, possible data exchange methods, security and privacy considerations and personal impacts.
- 3 How safe is your personal information?
 - a Review the definition of personal information on the Office of the Australian Information Commissioner website.
 - b List three websites that you have provided with personal information.
 - c Review the cybersecurity survey results at the link.
 - d Write a letter to the Office of the Australian Information Commissioner outlining three concerns you have with regard to your personal information.



4.1.10
Australian
Government
public data
policy system
4.1.11 What
is personal
information?
4.1.12 Data
breach
incidents,
causes and
response

4.2

DEVELOPING DATA EXCHANGE SOLUTIONS

In this section, you will analyse a data exchange problem and investigate possible approaches for data exchange over a network. You will develop data interfaces and algorithms that can be used in data exchange solutions and generate early prototypes of digital solutions that exchange data.

This section demonstrates how the Digital Solutions problem-solving process can be applied to data exchange problems, using the OpenActive project as an example. The OpenActive project has made a number of activity datasets available online, but as yet not many data consumer services are using them. The vision of the project was to create an open data environment to encourage data exchange solutions similar to those that exist in the tourism industry, where websites collate data from lots of different hotels and airlines to show travellers the best deal. The goal of the OpenActive project is to encourage people to be more active by making it easier for them to find nearby activity sessions that suit their interests and lifestyle.

To develop ideas about the possible solutions that could use OpenActive datasets, the develop and evaluate phases of the problem-solving process can be used.



Develop

Divergent thinking and mind maps can be used to brainstorm the problem. Convergent thinking can be applied to determine requirements and prescribed criteria for the solution. A concept map can identify components for a possible solution, and this can be refined to create formal diagrams such as DFDs, relational schemas, IPO charts and algorithms.



Evaluate

The solution can be analysed, tested and refined at each stage using checklists related to relevant prescribed and self-determined criteria. Systems thinking can be used to test use cases, and data structures and desk checks help to identify errors and ambiguity in algorithms.



Explore

To create early prototype solutions using code libraries and data exchange frameworks, the Explore and Generate phases of the problem-solving process are used.

Programming environments, code examples and tools can be sourced online and used or adapted for a data exchange project.



Generate

Algorithms and code are generated using computational thinking to create programmed components of the solution that request data from a data provider and present it to the data consumer.

Analysing a data exchange problem

Using design thinking to brainstorm a problem is a great way to begin thinking about the possibilities and challenges associated with a new project. Mind maps are a fast and effective way of collecting a range of divergent ideas, and can be completed individually or in a group to analyse a digital problem. As shown in figure 4.2.1, there are a number of considerations relevant to the OpenActive data exchange problem.

The right-hand side of the mind map explores the server side of the problem. There are a number of different fitness activity providers that have created OpenActive APIs. Each of these providers has interpreted the W3C OpenActive standard in a slightly different way. Some have created an array of sessions available from a single location, and others have created nested data structures that identify lots of different types of activities that take place at multiple addresses. The type of data available does not include any personal details, so there are no security or personal issues to consider. The flexibility of the OpenActive model makes it easy for activity providers to create APIs for their data; however, this creates complications for developers who want to aggregate the data as a consumer service using a standard format.

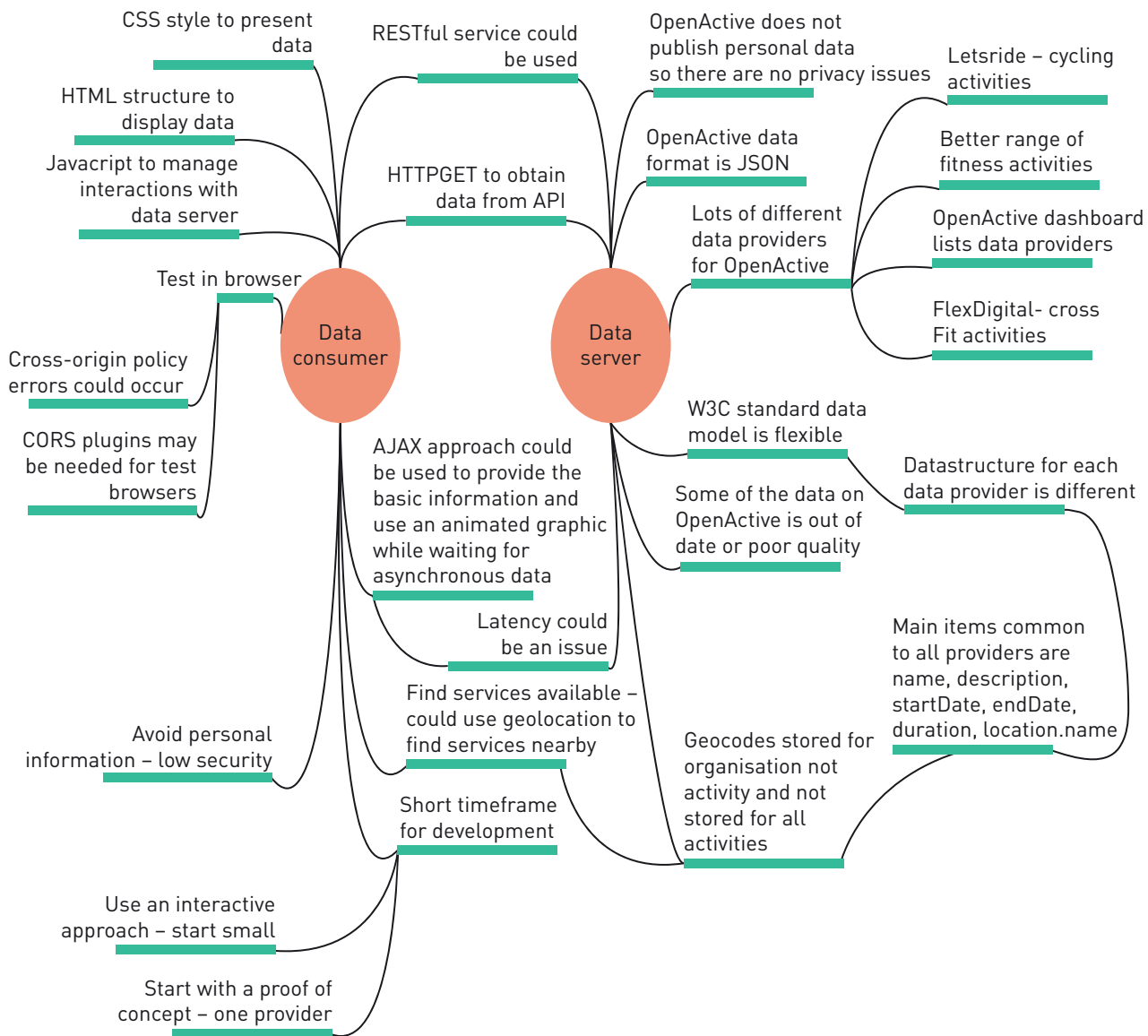


Figure 4.2.1 Brainstorming mind map for OpenActive data exchange

The left-hand side of the mind map explores the client side of the problem. Decisions need to be made about what data items to display, how to structure the content using HTML, how to style content with CSS and the type of framework to use to access the data. All the data structures are JSON, and some datasets are large enough to create potential latency issues, so an asynchronous data exchange framework will probably be required. Because the website consuming the data is in a different web domain to the website providing the

GROUP ACTIVITY 4.2.1

- 1 Work in groups of four to expand the mind map in figure 4.2.1.
- 2 In pairs, choose one branch of the mind map in figure 4.2.1 and make this the centre of a new mind map. Brainstorm additional considerations for this aspect of the problem.
- 3 After 5–10 minutes, swap mind maps with the other pair and build on what they began. As a group, highlight ideas that could be used to specify the requirements of the solution.



data, same-origin errors may occur, so a CORS browser plug-in to enable cross-origin resource sharing could be helpful. Some of the datasets include geocodes that could be used to map sessions. It would be helpful to use the browser device geocode location as a filter to show nearby activities, but not all datasets include the geocode data required for this to work. Some of the datasets seem to have test data only, and no current activities, so the data exchange product created will need to be a proof of concept and will not be able to exclude older sessions as these are the only sessions available in the dataset.

A concept map will help to identify and analyse possible components and connections between the parts of a potential data exchange solution. Considering the user and data interactions from various perspectives helps to identify the processes required to exchange data successfully and plan how to manage errors that may occur at particular points. Figure 4.2.2 shows one possible approach that could be used to retrieve data from an API and present it to the user.

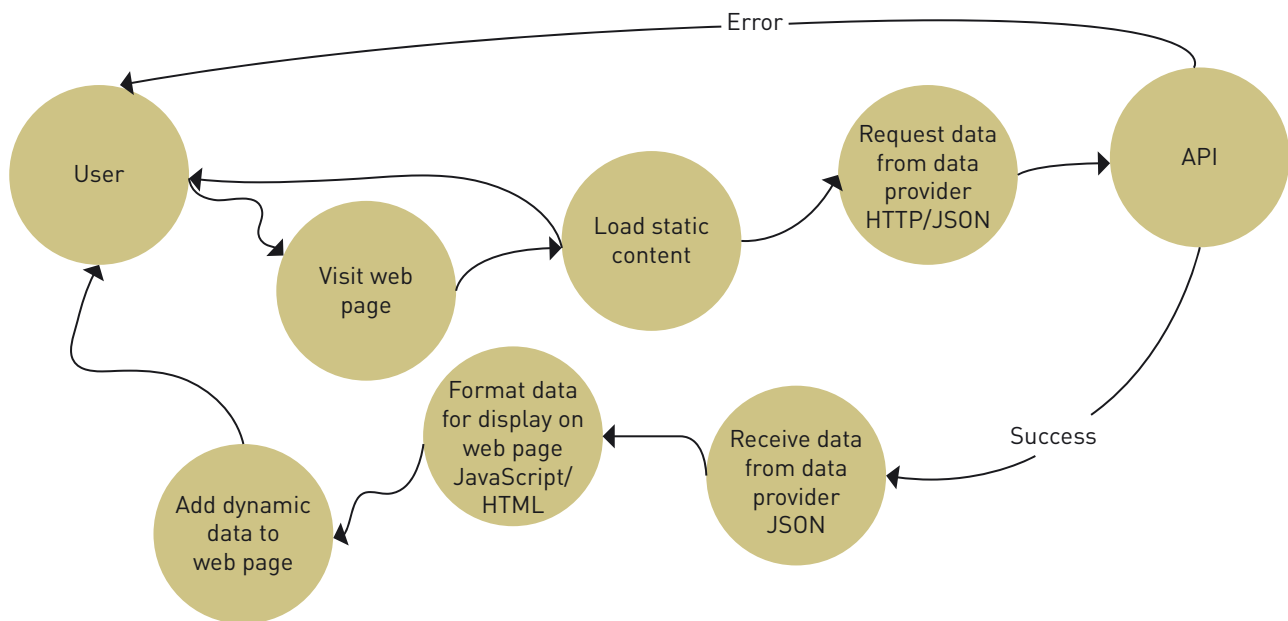


Figure 4.2.2 Concept map showing the user perspective of the OpenActive data exchange problem

ACTIVITY 4.2.1

Use systems thinking to analyse the concept map in figure 4.2.2 and understand the interactions in a data exchange solution.

- 1 Which of the input connections to the user concept will occur first?
- 2 Number the concepts in green that show the sunny day use case.
- 3 Number the concepts in red that show the rainy day use case.
- 4 Draw a system boundary around the components that could be affected by latency.

Once the connections between the different parts of the data exchange process have been analysed and refined, a formal diagram defining the requirements of the solution can be created. The data flow diagram in figure 4.2.3 represents the data flows that need to be considered in the OpenActive data exchange solution.

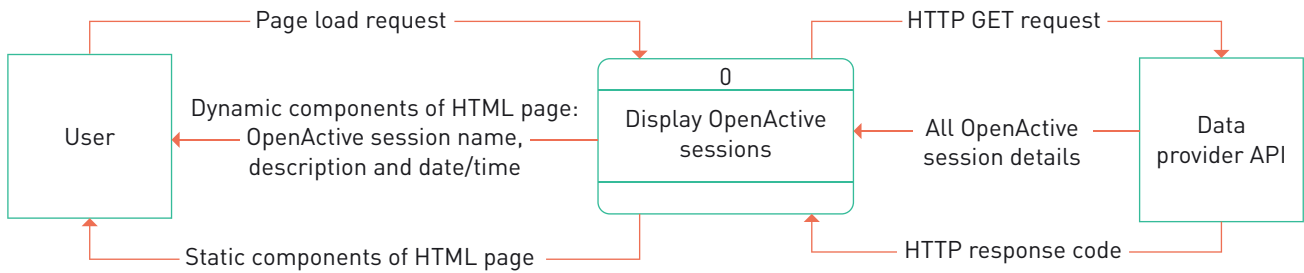


Figure 4.2.3 Top-level context DFD for OpenActive data exchange solution

When building a data exchange solution, it is best to apply an iterative approach. Start with the most important requirement and build up the solution by adding and testing one feature at a time. Some possible prescribed requirements of the OpenActive project are prioritised in the MoSCoW table (table 4.2.1).



Table 4.2.1 MoSCoW table showing prescribed requirements of the OpenActive project

MUST	Display data from at least one provider Organise the data using appropriate HTML structures and CSS styles
SHOULD	Display data from multiple providers Exclude past sessions based on the current date and time
COULD	Allow users to filter the data by location Allow users to filter data by provider Allow users to filter data by date/time e.g. weekday, am, pm
WON'T	Display all the data items from all data providers Update data or book sessions online Use geolocation data Require authentication

The prescribed criteria for version 1 of a solution are usually the most important or least troublesome features. In the OpenActive example the prescribed criteria for the first prototype are the MUST priorities from the MoSCoW table shown in table 4.2.1. The self-determined criteria are that the solution is fast and simple, as these were the overall aims of the OpenActive standard.

Developing the data components

The structure of the data components available to a data exchange solution is a specific format defined in API documentation. This API data structure will vary from provider to provider but, once published, the data structure for a specific API is usually fixed and cannot be changed.

The relational data structures defined in the OpenActive data model are **denormalised** or **serialised** for presentation in a JSON format. Each activity session record is represented as a JSON object in an indexed array. The JSON object includes key value pairs separated by commas. One-to-many data relationships (such as Organizer and Offer in the FlexDigital example below) are represented as nested JSON Objects. In a



relational database, this would be represented with a linking foreign key value, but in the JSON structure the entire record of key value pairs from the related table is duplicated for each record. Example data objects from two of the data providers are shown below:



4.2.1
Let's Ride

- **Let's Ride JSON data structure**

```
{
  "data": [
    {
      "id": 305,
      "title": "Mayor of London's Sky Ride",
      "start_time": "2010-09-05 09:30:00",
      "type": "City Ride",
      "suitable_for": "Children aged 8-15; Children aged 11-15; Adults 16+ only; Children aged 5-15",
      "Distance": "25.00",
      "Estimated_cycling_time": 0,
      "Estimated_total_time": 0,
      "city": "London",
      "Location": "",
      "difficulty": "Easygoing",
      "postcode": "SW1A 2BJ",
      "Start_latitude": "51.50303124",
      "Start_longitude": "-0.13020879",
      "description": "<p><img src='{image:}' alt=\"Cycling in your city\" \\/><br \\/>Whether you're saving money, considering the environment, avoiding the traffic jams, getting fit or wanting to take in some breathtaking scenery &ndash; let us point you in the right direction with cycling tips, clubs and routes in your local area.&nbsp; Cycle Hire, Cycle Superhighways, cycle parking, London's year of cycling, Biking Boroughs, cycle safety and training. For more information, <a href=\"http:\\\\www.london.gov.uk\\priorities\\transport\\cycling-re\",
      "Free_spaces": 63427,
      "Max_participants": null,
      "last_updated": "1970-01-01 01:00:01"
    }
  ]
}
```



4.2.2
FlexDigital

- **FlexDigital JSON data structure**

```
{
  "next": "\/oa?afterTimestamp=1530977993",
  "Items": [
    {
      "State": "updated",
      "kind": "Session",
      "Id": "59900e11f3a31ed46fc4cf38",
      "modified": "2017-08-13T08:30:09Z",
      "type": "Event",
      "image": "https:\\\\ludus-cover-photos.s3-eu-west-1.amazonaws.com\\/5911a42f6014e85e2d4612fa-26910\\/cover.jpg?1494331510786",
      "Organizer": {
        "Identifier": "5911a42f6014e85e2d4612fa",
        "name": "CrossFit Flexi",
        "logo": "https:\\\\ludus-cover-photos.s3-eu-west-1.amazonaws.com\\/5911a42f6014e85e2d4612fa\\/profile.jpg"
      },
      "Location": {
        "Geo": {
          "latitude": 53.7967173,
          "longitude": -2.3413003
        }
      },
      "startDate": "2017-08-27T08:30:00Z",
      "endDate": "2017-08-27T09:30:00Z",
      "eventStatus": "LIVE",
      "Offer": {
        "Price": 5,
        "priceCurrency": "GBP"
      },
      "remainingAttendeeCapacity": 20,
      "name": "Spartan Training",
      "description": "Being a Spartan is about being the best you can be. No matter your current fitness level, we're going to treat you like family and train you to be your own Spartan.\\n\\nSpartan Training, will help you lose fat, gain lean muscle, improve endurance, increase strength and become more flexible.",
      "url": "https:\\\\flexiapp.co.uk\\/activity\\/59900e11f3a31ed46fc4cf38"
    }
  ]
}
```



Develop

In the example JSON data structures above, yellow highlighting has been used to analyse the data structures for Let's Ride and FlexDigital. The highlighted fields are common to both structures. This shows that, although the data records are both about activity sessions, only two field names are common to each

dataset – ‘type’ and ‘description’. Both datasets also store starting date and time but they use different field names. Location field is available in both datasets but is empty in one. The description field for each dataset is very long, and some parts of the description have been truncated in these examples.

Reflecting on the prescribed and self-determined criteria of the data exchange solution, the most useful fields presented to the web page would probably be the title or name of the session, the date and time it is scheduled for and the description of the session (see table 4.2.2).



Table 4.2.2 The priority of the criteria influences which features are implemented earlier and which are added in later versions.

IDEA/POSSIBLE APPROACH	RELEVANT CRITERIA	RECOMMENDATION
Display OpenActive data fields common to all providers: • type and description • possibly date/time	MUST exchange data from one provider: yes COULD exchange data from multiple providers: yes SIMPLE: yes	Version 1 • Display data from one provider as a proof of concept • Identify provider name as static data • Include activity name, date/time, description as dynamic data Version 2 • Display data from multiple providers using CSV list of providers • Identify provider name as dynamic data • Include activity name, date/time, description as dynamic data • Group dynamic session data with dynamic provider name

Developing the data interface

The structure of the data interface will be determined by the developer of the data consumer application in consultation with potential end users. In some situations, the data consumer will be another process or device. In the case of the OpenActive data exchange problem, the data consumer is the web page that will be presented to the user.

The OpenActive data interface is constrained by the requirements listed in table 4.2.3.

Table 4.2.3 Selected requirements of the OpenActive project

MUST	Organise the data using appropriate HTML structures and CSS styles
WON'T	Require authentication

This means that no login process is required to access the data (WON'T require authentication), and HTML and CSS must be used to present the data retrieved from the API.

The data interface will include a combination of static and dynamic elements. The static elements will be the parts of the data interface that remain the same regardless of the data exchanged. These include items such as the title and information about the data provider and purposes for the data exchange solution. The dynamic elements are the parts of the data interface that will change depending on the number and value of data items exchanged. Static and dynamic elements of the OpenActive data exchange interface are represented in figure 4.2.4.

The sketch shown in figure 4.2.4 shows how the data from multiple providers could be presented. However, using the list of prioritised requirements to evaluate the sketch shows that the main requirement of the prototype is a proof of concept with only one data provider.



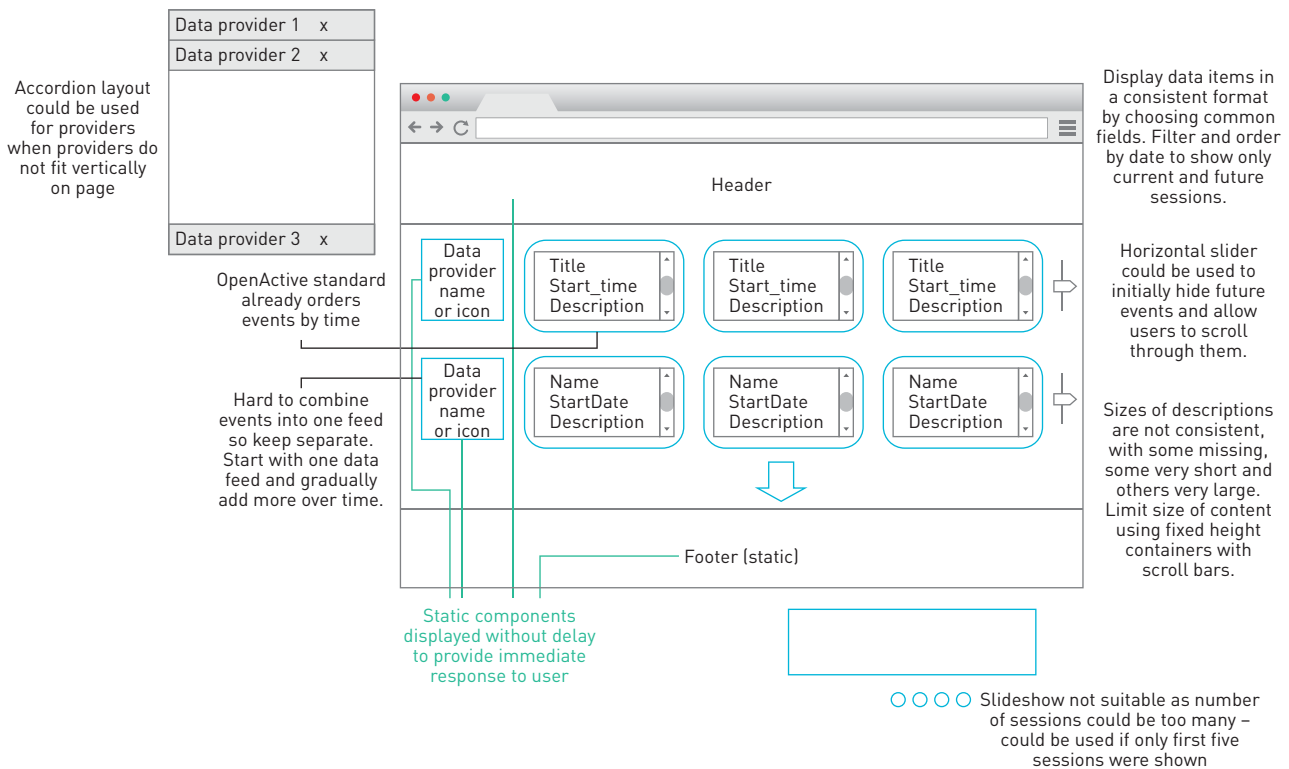


Figure 4.2.4 Static and dynamic elements of the OpenActive data exchange interface

Table 4.2.4 Criteria should be considered when evaluating ideas and approaches.

IDEA/POSSIBLE APPROACH	RELEVANT CRITERIA	RECOMMENDATION
Display retrieved data in provider groupings using features such as accordions and sliders that present a few records at a time	MUST organise data using HTML and CSS: yes SIMPLE: no	Version 1 Simple display with data from one provider. No need for sliders or accordions. Version 2 Display data from multiple providers. Use sliders or accordions to allow scrolled view of data.

Convergent thinking to align ideas to the prescribed criteria can be used to identify the most important priorities for version 1 and record useful ideas that could be introduced in later versions. This versioning process is fluid. Features initially deferred for version 2 may be brought forward into version 1 if time allows, or deferred until a potential version 3.

Managing presentation of dynamic data components

In the OpenActive data exchange problem, the number of records returned by the data provider is unknown. A flexible data presentation structure is required to accommodate some static components and an unknown number of dynamic components. Some common methods to manage presentation of dynamic data are tables, grids, accordions, sliders and scrolling containers.

The simplest approach for the first version of the prototype would be to use the CSS-grid format to display the data from the FlexDigital website on a single web page. Additional features and data providers can be added in later versions if time allows.

Grids

CSS grids allow the developer to dump data cells in HTML <div> containers and then organise them into rows and columns using CSS. The CSS Tricks website explains how CSS grids can be set up quickly and easily.

For example, the CSS code in the example below will set up a grid of cells that are 100 pixels wide with a gap of 5 pixels between each cell.

```
.currencyGrid {
  display: grid;
  grid-gap: 5px;
  grid-template-columns: repeat(auto-fit, minmax(100px, 1fr));
}
```

Tables

An HTML table structure can be used to dynamically present data. The HTML content fields are built and populated with dynamic data as it is received. Some examples of HTML table structures and formatting options are explained on the W3Schools website.

Accordions

An 'accordion' structure allows the developer to display large volumes of data efficiently. Data can be hidden or revealed by clicking on the title bars used to organise content into boxes within the accordion. A good tutorial and demonstration of the accordion layout is available at Codrops.

Sliders and scrolling containers

There are numerous online tutorials explaining how to use HTML, CSS, JavaScript and JQuery to create dynamic containers that scroll vertically or horizontally. An example of the code required to build a slider with JavaScript is explained on the W3Schools website.

Developing the programmed components

Programmed components in a data exchange solution control the process and timing of the data exchange. Where the structure of the data interface does not match the structure of the underlying API data, programmed components must also be developed to transform the data into a suitable presentation format.

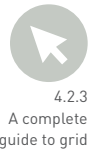
Early brainstorming for the OpenActive project identified a need to use an asynchronous approach for data exchange to address potential latency issues. In order to implement an asynchronous data exchange, the interactions between the process and the data provider will need to be programmed as a two-step process. Firstly, the request is sent (1) and a separate function will be triggered to format the data when it is returned (2). This is shown in the DFD in figure 4.2.5.

The data flows shown on the DFD in figure 4.2.5 include:

- **page load request** – the HTTP request from the user to load the page
- **static components of the HTML page** – the HTML structures that will appear immediately
- **dynamic components of the HTML page** – the HTML structures that will be created when the data is available
- **HTTP GET request** – a correctly formatted request for data. The OpenActive standard proposes the following format for a request:

```
GET /api/rpde/sessions?afterTimestamp=1453931101&afterId=%7Bc15814e5-8931-470c-8a16-ef45afedaece%7D
```

- **HTTP response code** – a code that communicates the status of the request. The OpenActive standard also identifies the possible values of an HTTP response or status code, as shown in table 4.2.5.



4.2.3
A complete
guide to grid



4.2.4
HTML
tables

4.2.5

Accordion
with CSS3

4.2.6

W3 CSS
slideshow

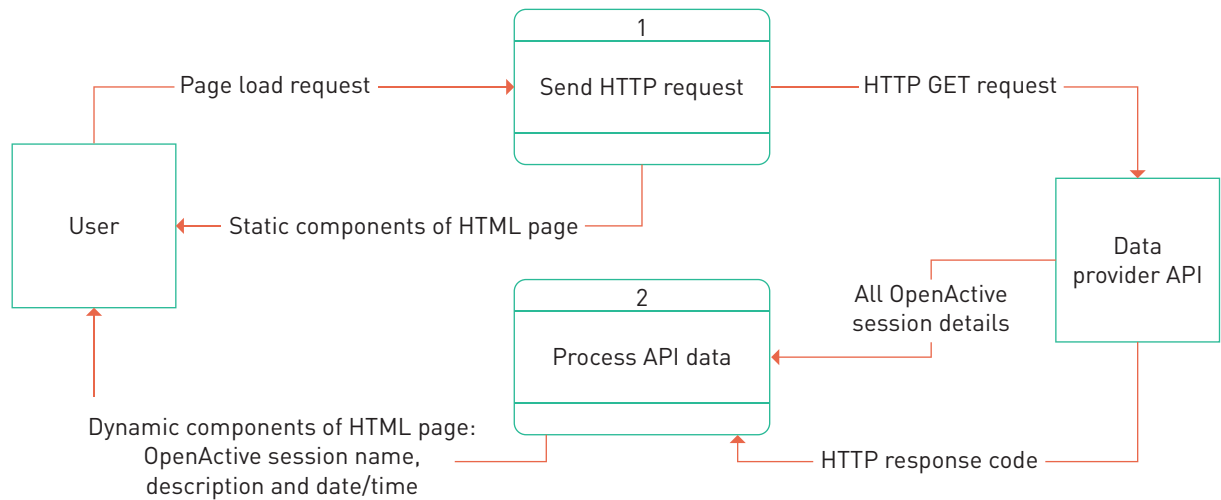


Figure 4.2.5 Level 1 data flow diagram for asynchronous data exchange

Table 4.2.5 The OpenActive standard defines how the data consumer application should respond on receipt of specific HTTP response codes.

HTTP STATUS CODE	INTERPRETATION	EXPECTED CONSUMER (SYSTEM 2) BEHAVIOUR
200	Normal response	Continue download/infrequent polling
503	Temporary overloading or maintenance of the server	Retry after a random interval between 60 and 120 minutes, to ease load from multiple consumers
404 or 410	Endpoint not found at this location	Consider this endpoint in an error state and do not retry

- OpenActive session details** – the response data returned as a result of the HTTP GET request. The formats for OpenActive session details were explored earlier using the JSON data structures for FlexDigital and Let's Ride. The OpenActive standard provides guidelines for the data format of requests and responses. Here is an example response:

```

{
  "next": "http://www.example.com/api/rpde/sessions?afterTimestamp=1453931925&afterId=%7Bd97f73fb-4718-48ee-a6a9-9c7d717ebd85%7D",
  "items": [
    {
      "state": "updated",
      "kind": "session",
      "id": "{c15814e5-8931-470c-8a16-ef45afedaece}",
      "modified": 1453931101,
      "data": {
        "@context": "https://www.openactive.io/ns/oa.jsonld",
        "type": "Event",
        "identifier": "session-01jz93i3k1p3",
        "superEvent": {
          "identifier": "eventType-ak6zw8ntsiqq"
        },
        "startDate": "2016-05-09T18:15:00Z",
        "endDate": "2016-05-09T19:15:00Z",
        "remainingAttendeeCapacity": 0,
        "url": "https://bookingsystem.com/hulahoop/e/ev-ssyp-20160509191500?r=oa",
      }
    }
  ]
}
  
```

```

    "attendanceCount": 29,
    "location": {
      "type": "Place",
      "identifier": "location-jbnentuhxsnp",
      "url": "https://leisurecentre.org.uk/leisure-centre/camden/kentish-
town-sports-centre",
      "name": "Kentish Town Sports Centre",
      "address": {
        "type": "PostalAddress",
        "addressLocality": "Camden",
        "addressRegion": "London West",
        "postalCode": "NW5 3DU",
        "streetAddress": "Kentish Town Sports Centre"
      }
    }
  },
  {
    "state": "deleted",
    "kind": "session",
    "id": "{d97f73fb-4718-48ee-a6a9-9c7d717ebd85}",
    "modified": 1453931925
  }
],
"license": "https://creativecommons.org/licenses/by/4.0/"
}

```

Each of the processes in the data flow diagram need to be generated as programmed components, and the data flows into and out of the process will become the inputs and outputs for the process.

Investigating programming environments

One of the most popular code libraries used for JSON data exchanges is jQuery.

The `jQuery.getJSON` method loads data from a given server using a HTTP GET request. This is a shorthand AJAX method within the jQuery library. To use jQuery methods, it will be necessary to download the jQuery library and save it in a local web folder, available from the web page.

The `getJSON` method documentation suggests the following code sample (figure 4.2.6), which can be adapted for the OpenActive solution.

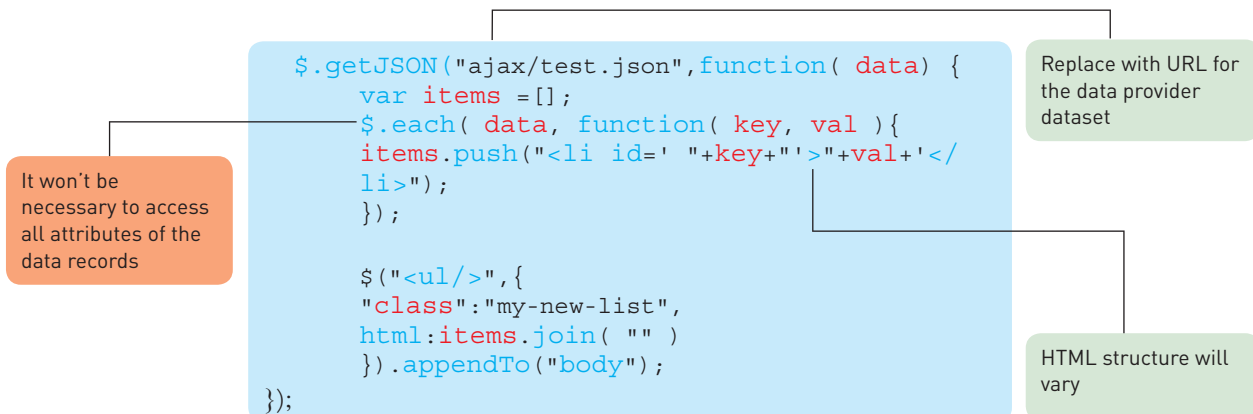
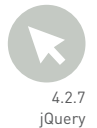


Figure 4.2.6 Annotated code sample



A number of data exchange code libraries are available online and can be used to construct the generic structure of data exchange components. Whenever an external code sample or library is used, programming conventions require the developer to identify the source of any borrowed code as comments within the code. The main processes that will be unique to each solution will be the functions that manage the transformation of data from the API data format to the presentation format defined by the data interface.

Developing data exchange algorithms

Generation of programmed components for any digital solution always begins with an algorithm. An IPO chart is a useful tool to ensure that all possible inputs and required outputs are considered by the algorithm. Pseudocode for each process identified on the data flow diagram also needs to be developed.

There are two distinct processes required in the OpenActive data exchange solution identified on the DFD level 1 diagram in figure 4.2.5 as 'Send HTTP request' and 'Process API data'. The inputs, algorithm process and outputs for these processes are described in the IPO charts below.



Table 4.2.6 IPO chart for 'Send HTTP request'

Inputs	Process	Outputs
Event: Page load request	Display static elements of the web page Send HTTP GET request with the callback function as 'Process API data'	HTTP GET request Static HTML

Table 4.2.7 IPO chart for 'Process API data'

Inputs	Process	Outputs
OpenActiveSessions (List of OpenActive session details in JSON format - varies for each data provider) HTTPResponse (HTTP response code: success=200, error=404 or 410, busy=503)	<pre> IF HTTPResponse = success THEN REPEAT Get next OpenActiveSession OUTPUT opening <div> tag OUTPUT session name inside <h3> tags OUTPUT date/time inside <p> tags OUTPUT description inside <p> tags OUTPUT closing <div> tag UNTIL last session in OpenActive-Sessions processed. ELSE IF HTTPResponse = error THEN OUTPUT "Data not available" ELSE OUTPUT "Data provider busy - try again in a couple of hours" ENDIF ENDIF WRITE OUTPUT to web page </pre>	List of session name, date/time, description in HTML format

The algorithm for 'Process API data' has a nested structure, as shown in figure 4.2.7.

Refining algorithms to remove ambiguity is particularly important when developers work in a team. The developer that writes the code is not always the same developer who wrote the algorithm.



```
IF HTTPResponse = success THEN
```

```
  REPEAT
    Get next OpenActiveSession
    OUTPUT data
  UNTIL last session in OpenActiveSessions processed
```

```
ELSE
```

```
  IF HTTPResponse = error THEN
    OUTPUT data
  ELSE
    OUTPUT "Data provider busy - try again in a couple
    of hours"
  ENDIF
```

```
ENDIF
```

```
WRITE OUTPUT to webpage
```

This REPEAT block is nested inside the true condition of the IF THEN ELSE block. This means that it will only run if the IF condition is true.

This IF THEN ELSE block is nested inside the false condition of the IF THEN ELSE block. This means that it will only run if the IF condition is false.

Figure 4.2.7 Complex algorithms can be constructed by nesting one control structure within another control structure.

According to the OpenActive standard, there are two possible response codes to indicate an error, 404 or 410. To remove ambiguity from the nested IF block in figure 4.2.7, the condition:

***HTTPResponse* = error**

can be rewritten as the more specific compound condition:

(*HTTPResponse* = 404 OR *HTTPResponse* = 410)



Evaluate

ACTIVITY 4.2.2

- 1 Evaluate the algorithm in table 4.2.7, the 'Process API data' IPO chart, by completing a desk check with different HTTP response codes and the following OpenActiveSessions data:

```
{ {"name": "Spartan Training", "startDate", "2017-08-27T08:30:00Z", "Description", "Being a Spartan is about being the best you can be. No matter your current fitness level, we're going to treat you like family and train you to be your own Spartan."}, {"name": "H.I.I.T it Hard", "startDate": "2017-08-28T18:00:00Z", "Description": "High Intensity Interval Training- small group class combines elements of cardio, strength training, and speed and agility drills."}, {"name": "Relentless", "startDate": "2017-08-29T18:00:00Z", "Description": "An 'out of the box' class, using kettle-bells, ropes, tires, weights, sledge hammers, plus lots more.... "}, {"name": "H.I.I.T it Hard", "startDate", "2017-08-30T18:00:00Z", "Description": "High Intensity Interval Training- small group class combines elements of cardio, strength training, and speed and agility drills."} }
```

DESK CHECK	VARIABLE NAME	TEST VALUE
1	HTTPResponse	200
2	HTTPResponse	404
3	HTTPResponse	503
4	HTTPResponse	403

- 2 How could you improve this algorithm to format date/times more effectively?
- 3 What assumptions does the algorithm make?
- 4 How could you improve it to make it more robust?



Evaluate



Selecting an appropriate data exchange approach

There are a number of different approaches that could be used to request the data from FlexDigital. Since the data is already in JSON format, the simplest approach was to use the jQuery method. If the data was in a different format, XMLHttpRequest could be used.

- **AJAX with jQuery (JSON only)**

```
//request JSON data
jQuery.getJSON(requestURL,function(data) {

    //do something with the data
})
```

- **AJAX with XMLHttpRequest (JSON or XML)**

```
//Set up a request object
var request = new XMLHttpRequest();

//Open a new request
request.open('GET', requestURL);

//Set the request type
request.responseType = 'json';

//Send the request
request.send();

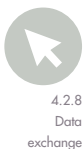
//wait for response
request.onload = function() {
    var data = request.response;

    //do something with the data
})
```

ACTIVITY 4.2.3

- 1 Implement the code in the table below to display data from a currency exchange website. Note: You will need to obtain a free API key from the fixer.io website to use in the bolded section of the example code.

INDEX.HTML	STYLE.CSS	DATA.JS
<pre><!DOCTYPE html> <html> <head> <meta charset="utf-8"> <title>Currency Grid</title> <link rel="stylesheet" href="style.css"> <script src="jquery-3.2.1.min. js"></script> <script src="data.js"></script> </head> <body> <header> <!-- This section will display the title to show the selected country and the current date --> </header> <section class="currencyGrid"></pre>	<pre>.currencyGrid { display: grid; grid-gap: 5px; grid-template- columns: repeat(auto-fit, minmax(100px, 1fr)); grid-template- rows: repeat(2, 100px); }</pre>	<pre>var header = document. querySelector('header'); var section = document. querySelector('section'); //Store the URL of the json in a variable var requestURL = http://data.fixer.io/ api/latest?access_key=[enter your access key here] //Get the exchange rate data for the section jQuery. getJSON(requestURL,function(data){ //Set up the header var output=""; output += "<h1>"+ "data.base" + "</h1>"; output += "<h2> Currency rates for " + data.date + "</h2>"; header.innerHTML=output;</pre>



INDEX.HTML	STYLE.CSS	DATA.JS
<pre> <!-- This section will display a responsive grid containing different currency conversion rates applicable to the selected country's currency --> </section> </body> </html> </pre>		<pre> //Set up the divs containing exchange data from the rates array output=""; for (x in data.rates) { output += "<div>"; output += "<h3>" + x + "</h3>"; output += "<p>" + data.rates[x] + "</p>"; output += "</div>"; } section.innerHTML = output; }); </pre>

- 2 Make changes to the CSS file to display the data in different formats.
- 3 Make changes to data.js to access the data using XMLHttpRequest instead of getJSON.

The XMLHttpRequest method of interacting with a JSON or XML server is limited by a same-origin policy. This is a security feature of the HTTP protocol that blocks interactions from external requests of unknown origin as a means of protecting the server from a malicious attack. When sharing data publicly, the same-origin policy will often cause cross-platform origin errors which block the HTTP GET request and prevent the data from being shared as intended. Many developers work around this problem by using JSONP (JSON with Padding), which does not rely on the HTTP GET method or the cross-origin resource sharing (CORS) framework supported by most modern browsers.



Once the algorithms for the solution have been unambiguously defined and the data structures have been identified and mapped to a data interface, the creation of code for the solution is relatively straightforward. Building the solution gradually, and testing after each change, is usually faster than looking for multiple errors across hundreds of lines of code.

Code to interact with the DigitalFlex open data website is outlined below:

- **HTML**

```

<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Open Active Sessions</title>
    <link rel="stylesheet" href="style.css">
  <script src="jquery-3.2.1.min.js"></script> //update to match your jquery
                                           //library version
    <script src="flexDigital.js"></script>
  </head>

  <body>
    <header>
      <h1>FlexiDigital</h1>
    </header>

    <section class="activitySessionGrid">
<!-- This section will display a responsive grid containing activity
sessions available -->
    </section>
  </body>
</html>

```





- **CSS**

```
body{
background-color:#C0C0C0;
text-align: center;
}

div {
    background-color:white;
    padding:5px;
    border:1px;
    border-radius:25px;
}

.activitySessionGrid {
    display: grid;
    grid-gap: 20px;
    grid-template-columns: repeat(auto-fit, minmax(250px, 1fr));
}
```

- **JavaScript**

```
//Set the url to the FlexDigital API
var url ='https://open.flexidigital.co.uk/oa/';

//Send a request for the data
jQuery.getJSON(url,function(data){
    /*****
    Define the callback function
    This function will be used to process the HTTP Response
    Purpose: To display the activity session data in the appropriate section
    *****/
    //Set up a pointer to the section of the page
    //where the dynamic data will be added
    var section = document.querySelector('section');

    //Initialise the output string that will be used to build the HTML
    var output="";

    //Inspect each key:value pair in the JSON data structure
    for (x in data.items){
        //Append the HTML to define the container object
        output+="<div>";
        //Append the HTML to display the name of the activity
        output+="<h3>"+data.items[x].name+"</h3>";
        //Append the HTML to display the date and time for the activity
        output += "<p><strong>"+data.items[x].startDate+"</strong><br>";
        //Append the HTML to display the description of the activity
        output += "<p>"+data.items[x].description+"<br>";
        output += "</div>";
    }

    //Add the HTML to the webpage
    section.innerHTML = output;
})
```

FlexDigital				Flex Digital
Spartan Training 2017-08-27T08:30:00Z Being a Spartan is about being the best you can be. No matter your current fitness level, we're going to treat you like family and train you to be your own Spartan. Spartan Training will help you lose fat, gain lean muscle, improve endurance, increase strength and become more flexible.	H.I.I.T It Hard 2017-08-28T18:00:00Z The fun, fast-paced -High Intensity Interval Training- small group class combines elements of cardio, strength training, and speed and agility drills.	Relentless 2017-08-29T18:00:00Z These are described as an 'out of the box' class, using kettle-bells, ropes, tires, weights, sledge hammers, plus lots more..... Basic gymnastic holds, from simple to advanced movements, all done under a resistance with a variety of equipment, we pull, throw, slam, hit, and push things. Making you love every minute, vent the weeks frustration by hitting a Tyre with a sledge hammer, sounds fun right! Its awesome..... You will build strength, Stamina, fitness, muscle and lose weight.	H.I.I.T It Hard 2017-08-30T18:00:00Z The fun, fast-paced -High Intensity Interval Training- small group class combines elements of cardio, strength training, and speed and agility drills.	
Relentless 2017-08-31T18:00:00Z These are described as an 'out of the box' class, using kettle-bells, ropes, tires, weights, sledge hammers, plus lots more..... Basic gymnastic holds, from simple to advanced movements, all done under a resistance with a variety of equipment, we pull, throw, slam, hit, and push things. Making you love every minute, vent the weeks frustration by hitting a Tyre with a sledge hammer, sounds fun right! Its awesome..... You will build strength, Stamina, fitness, muscle and lose weight.	Warrior Games 2017-09-02T08:00:00Z Warrior Games is performed in Teams of 2 or 3 records to complete a Fitness task	FREE Crossfit Taster Session 2017-09-02T09:00:00Z The classes are all 1 hour long and will have a warm up, a skill of strength section and	Spartan Training 2017-09-03T08:30:00Z Being a Spartan is about being the best you can be. No matter your current fitness level, we're going to treat you like family and train you to be your own Spartan. Spartan Training will help you lose fat, gain lean muscle, improve endurance, increase strength and become more flexible.	

Figure 4.2.8 Screen capture of the OpenActive session application, showing all the session data for one data provider

TEST YOUR KNOWLEDGE

- Sketch the four phases of the Digital Solutions problem-solving process and annotate the sketch to show the type of activities that occur during each phase.
- What types of features should be included in version 1 of a prototype solution?
- What should be used to regularly evaluate ideas and components of a prototype data exchange solution?
- Why was an asynchronous approach used for the OpenActive solution discussed in this section?
- Describe three methods of managing the presentation of large volumes of data.
- What HTTP response code is returned for a successful HTTP GET request?
- What does 'HTTP error 503' mean?
- Explain what is meant by a nested data structure. Give an example from the OpenActive project.
- What is the relationship between data flows in a data flow diagram and an IPO chart?
- Explain what is meant by a nested control structure (algorithms) and give an example using appropriately indented pseudocode.
- Why is it important to remove ambiguity from algorithms?
- AJAX can be used with JSON and XML data. Can the jQuery function `getJSON()` also be used to exchange data in both formats?
- Explain what is meant by 'cross-platform origin errors'.

APPLY YOUR KNOWLEDGE

- Research two data exchange formats not explained in the text.
- Ask Izzy is an Australian application that uses open data to connect homeless people with essential services.
 - Use the Internet to research the Ask Izzy data exchange application.
 - Sketch a mind map identifying technology, ideas and issues that relate to the Ask Izzy data exchange application.
- Frugl is a website that allows customers to compare grocery prices from different local supermarkets and display them in a web browser. Sketch a data interface that could be used in a similar data exchange application for an Internet fridge. The application would regularly check the availability of essential items and automatically order items below preferred stock levels from the store with the lowest price.
- Create a data exchange solution that presents data from the OpenActive Better URI.



4.3

DEVELOPING AND GENERATING METHODS OF DATA SECURITY

In previous sections, we have explored the concepts and processes relating to exchanging information via networks. This has included data such as open data, which does not require protection and can be exchanged freely. Other forms of data, however, such as personal information, financial accounts and sensitive services, require high levels of security to protect an individual's right to privacy and to stop unauthorised access that could lead to negative impacts such as financial or identity theft, overt surveillance and user profiling.

In this section, we will look at how sensitive data can be protected by developing, generating and testing some of the encryption and authentication strategies appropriate for securing data for transmission and exchange. We will recognise and describe how encryption and hashing techniques are used in the storage, transfer and checking of data.



Figure 4.3.1 The invention of the Enigma machine increased the complexity of ciphers and codes.

Keeping secrets (across open networks)

Scrambling information to keep it secret is not a new idea. The art of encryption and decryption, commonly referred to as cryptography, has been found to have begun thousands of years ago in ancient Egyptian, Greek and Roman times. In the 20th century, the complexity of using ciphers and codes increased dramatically with the invention of complex mechanical devices such as the Enigma machine, and ultimately through computing hardware.

History of the Caesar cipher

The Caesar cipher is one of the earliest examples of making information secret through the use of encryption and decryption. It is said to be named after Roman general and politician Julius Caesar, who was said to have used it in the 1st century BC to protect messages to and from his military leaders. It was said that if Caesar needed to pass any message that was confidential, he employed the use of his cipher to protect it from prying eyes when being carried or transferred. Historians believe that Caesar used other variations and more advanced examples of ciphers throughout his reign and the concept of encrypting and decrypting confidential messages was passed to his successors.

From a computing perspective, the Caesar cipher is simply an algorithm, a set of instructions that can be computed to either encrypt or decrypt a message. It works by shifting letters of the alphabet using a wrap-around method (figure 4.3.3). For each letter in an original message, a replacement letter is used, resulting in a scrambled message.

At the receiving end, the opposite occurs. The shift is applied in reverse, the encrypted letters revert back to the original letters, and the message is unscrambled to its original state. The cipher relies on the sender and the receiver both knowing the shift key. The concept of a key is central to all ciphers, both simple and complex.



Getty Images/Hulton Archive

Figure 4.3.2 Julius Caesar used the Caesar cipher to protect messages of military significance.

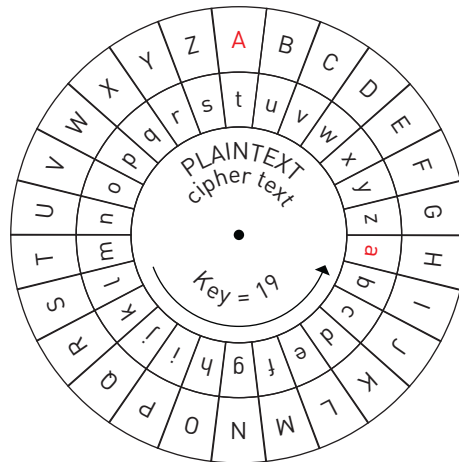


Figure 4.3.3 The Caesar shift

A practical example of encryption and decryption using the Caesar cipher is shown below, using a shift key of 23:

- **Plaintext**
THE QUICK BROWN FOX JUMPS OVER THE LAZY DOG
- **Plaintext + 23 = cipher text**
QEB NRFZH YOLTK CLU GRJMB LSBO QEB IXWV ALD
- **Cipher text – 23 = plaintext**
THE QUICK BROWN FOX JUMPS OVER THE LAZY DOG

As shown, the shift of the letters using a specific key scrambles and unscrambles the message as intended.

Dec	Hex	Oct	Chr	Dec	Hex	Oct	HTML Chr	Dec	Hex	Oct	HTML Chr	Dec	Hex	Oct	HTML Chr
0	0	000	NULL	32	20	040	 Space	64	40	100	@ @	96	60	140	` `
1	1	001	Start of header	33	21	041	! !	65	41	101	A A	97	61	141	a a
2	2	002	Start of text	34	22	042	" "	66	42	102	B B	98	62	142	b b
3	3	003	End of text	35	23	043	# #	67	43	103	C C	99	63	143	c c
4	4	004	End of transmission	36	24	044	$ \$	68	44	104	D D	100	64	144	d d
5	5	005	Enquiry	37	25	045	% %	69	45	105	E E	101	65	145	e e
6	6	006	Acknowledgement	38	26	046	& &	70	46	106	F F	102	66	146	f f
7	7	007	Bell	39	27	047	' <>	71	47	107	G G	103	67	147	g g
8	8	010	Backspace	40	28	050	({	72	48	110	H H	104	68	150	h h
9	9	011	Horizontal tab	41	29	051	) }	73	49	111	I I	105	69	151	i i
10	A	012	Line feed	42	2A	052	* *	74	4A	112	J J	106	6A	152	j j
11	B	013	Vertical tab	43	2B	053	+ <+>	75	4B	113	K K	107	6B	153	k k
12	C	014	Form feed	44	2C	054	, ,	76	4C	114	L L	108	6C	154	l l
13	D	015	Carriage return	45	2D	055	- <->	77	4D	115	M M	109	6D	155	m m
14	E	016	Shift out	46	2E	056	. .	78	4E	116	N N	110	6E	156	n n
15	F	017	Shift in	47	2F	057	/ </>	79	4F	117	O O	111	6F	157	o o
16	10	020	Data link escape	48	30	060	0 0	80	50	120	P P	112	70	160	p p
17	11	021	Device control 1	49	31	061	1 1	81	51	121	Q Q	113	71	161	q q
18	12	022	Device control 2	50	32	062	2 2	82	52	122	R R	114	72	162	r r
19	13	023	Device control 3	51	33	063	3 3	83	53	123	S S	115	73	163	s s
20	14	024	Device control 4	52	34	064	4 4	84	54	124	T T	116	74	164	t t
21	15	025	Negative ack.	53	35	065	5 5	85	55	125	U U	117	75	165	u u
22	16	026	Synchronous idle	54	36	066	6 6	86	56	126	V V	118	76	166	v v
23	17	027	End of trans. Block	55	37	067	7 7	87	57	127	W W	119	77	167	w w
24	18	030	Cancel	56	38	070	8 8	88	58	130	X X	120	78	170	x x
25	19	031	End of medium	57	39	071	9 9	89	59	131	Y Y	121	79	171	y y
26	1A	032	Substitute	58	3A	072	: :	90	5A	132	Z Z	122	7A	172	z z
27	1B	033	Escape	59	3B	073	; ;	91	5B	133	[[	123	7B	173	{ {
28	1C	034	File separator	60	3C	074	< <	92	5C	134	\ \	124	7C	174	|
29	1D	035	Group separator	61	3D	075	= <=>	93	5D	135	]]	125	7D	175	} }
30	1E	036	Record separator	62	3E	076	> >	94	5E	136	^ ^	126	7E	176	~ ~
31	1F	037	Unit separator	63	3F	077	? ?	95	5F	137	_ _	127	7F	177	&#127; Del

Figure 4.3.4 The ASCII table



Making the Caesar cipher practical to computers

To make the Caesar cipher applicable to the computing world, it can be adapted to use character tables such as ASCII or UTF-8. These tables represent each identifiable computer character against a number. These numbers can be shifted to scramble characters, including letters, numbers and special characters. The full table or a subset of the table could be employed.

The algorithm would be as follows (using an IPO chart):

Input	Process	Output
Plaintext and keyshift	SET alphabet = abcdefghijklmnopqrstuvwxyz SET encryptedtext = "" FOR each letter in the Plaintext SHIFT letter by keyshift APPEND letter to encryptedtext; END FOR	encryptedtext

The code example (function) might look like this:

```
private string CaesarCipher(string inputText, int intShift)
{
    string alphabet = "abcdefghijklmnopqrstuvwxyz";
    string OutputText = "";
    foreach (char chrletter in inputText)
    {
        int charposition = alphabet.IndexOf(chrletter);
        int newcharposition = (charposition + intShift) % 26;
        OutputText += alphabet.Substring(newcharposition, 1);
    }
    return OutputText;
}
```

Breaking the cipher

The Caesar cipher is considered extremely weak and easily broken by a third party. This is due to the fact that there are limited variations, and a possible attacker could simply check each possible shift key to find a plausible solution. A marginally sophisticated code-breaking algorithm could unscramble these sorts of simple substitution approaches in microseconds.

As shown in table 4.3.1, with a simple desk check using a brute force technique on a scrambled message for the Caesar cipher, it is easy to identify an unscrambled segment of the true message.

Table 4.3.1 Breaking the Caesar cipher with a simple desk check

KEY SHIFT	MESSAGE SEGMENT
0	GLJLWDOVROXWLRQV
1	FKIKVCNUQNWVKQPU
2	EJHJUBMTPMVUJPOT
3	DIGITALSOLUTIONS
4	CHFHSZKRNKTSNMR
5	BGEGRYJQMJSRGMQLQ
...	...
24	INLNYFQXTQZYNTSX
25	HMKMXEPWSPYXMSRW

The key is discovered and the rest of the message can be decoded with ease.

Polyalphabetic ciphers

A polyalphabetic cipher is one in which the cipher alphabet changes in the course of the message. The polyalphabetic Trithemius cipher, for example, was an important step in the development of very secure ciphers. The Trithemius cipher, however, does not use a key, and is thus quite weak, as it encrypts every message the same way. A cipher that uses a key is more secure than one that does not. Among the many polyalphabetic ciphers that do use a key is the



Vigenère cipher. This combines a series of Caesar ciphers based on a secret keyword, and uses the idea of modulo (or mod) 26, in which the alphabet can be shifted in 26 different ways, one for each letter of the alphabet. The German Enigma machine, used to destroy Allied ships in World War II, is an example of an advanced polyalphabetic cipher, and was made famous in the movie *The Imitation Game*.

The polyalphabetic cipher generates numbers from a shift word (the key). It then uses these numbers to apply multiple shifts, rather than just one single shift. Let's look at an example.

- **Step 1**
Pick a keyword – for example, 'snake'. The longer the keyword, the more secure it will be.
- **Step 2**
Convert the keyword to numbers, using the position of each letter in the alphabet (starting from 0):
S = 18
N = 13
A = 0
K = 10
E = 4

- **Step 3**
This sequence of numbers (18, 13, 0, 10, 4) is then repeated across the message:

M	E	E	T		M	E		A	T		E	L	E	P	H	A	N	T		L	A	K	E
18	13	0	10		4	18		13	0		10	4	18	13	0	10	4	18		13	0	10	4

- **Step 4**
Now you can use a normal Caesar cipher to encrypt the message with the help of a shift key based on the number below each letter:
Move M 18 places = E
Move E 13 places = R
Move E 0 places = E
Move T 10 places = D
Move M 4 places = Q

- **Step 5**
Send the encrypted message.

- **Step 6**
The receiver can now decrypt the message by going backwards in the alphabet – that is, by subtracting the shift numbers from the position of each letter. To do this, of course, the receiver must have a copy of the secret word or 'shift word'.

The task is done – but wait! What if a code breaker was able to intercept the data packets being sent and establish the letter frequency (how many As, Bs, etc)? Remember, a code breaker looks for patterns (called information leaks). If a pattern is established, the keyword could be revealed. The advantage of using a shift word instead of a single shift key is that multiple shifts occur. In the case of 'SNAKE', five shifts occur. This means that instead of trying to identify a single pattern, the attacker would need to identify patterns in five different shifts. Quite a task. Do you think you are up to it?

In the movie *The Imitation Game*, a weather forecast was broadcast every morning at 6 a.m. over German public radio, enabling everyone, including the Allies, to track the message. The forecast contained a key, enabling German U-boat crews to decrypt Enigma messages that contained enemy ship co-ordinates so a surprise attack could take place. Cracking this code was vital to winning the war and, as told in the movie, the British scientist Alan Turing played a pivotal role in this. Once it became possible to crack intercepted coded messages, the Allies were able to defeat German U-boats in many crucial engagements, including the Battle of the Atlantic. It has been estimated that this work shortened the war in Europe by more than two years and saved over 14 million lives. Alan Turing was highly influential in the development of theoretical computer science. The Turing machine he developed can be considered a model of a general purpose computer, and he is widely regarded as the father of theoretical computer science and artificial intelligence.



ACTIVITY 4.3.1

Complete the rest of the cipher:

Move E 18 places =	Move A 10 places =
Move A 13 places =	Move N 4 places =
Move T 0 places =	Move T 18 places =
Move E 10 places =	Move L 13 places =
Move L 4 places =	Move A 0 places =
Move E 18 places =	Move K 10 places =
Move P 13 places =	Move E 4 places =
Move H 0 places =	

Here are some tips for cracking this cipher:

- 1 Search the cipher text for repeated strings of letters; the longer the strings you find, the better (for example, you might find the string 'KPQ' four times). Note where strings are by circling them or highlighting them in some manner.
- 2 For each occurrence of a repeated string, count how many letters are between the first letter in that string and the first letter in the next occurrence. Add one. So, for example, if our cipher text contains KPRE IJKO KPQAE, we count that there are nine letters between the K in the first KPQ and the K in the second KPQ; adding one yields 10.
- 3 Factor the number you got in the above computation. In our case, the factors of 10 are 2 and 5.
- 4 Repeat this process with each repeated string you find and make a table of common factors. The most common factor is probably the length of the keyword that was used to encrypt the cipher text. (In our case, let's assume it was 5.) Call this number 'n'.
- 5 Do a frequency count on the cipher text, on every nth letter. You should end up with n different frequency counts.
- 6 Compare these counts to standard frequency tables to figure out how much each letter was shifted by.
- 7 Undo the shifts and read the message.

Let's use an IPO chart to look at the example of a letter home.

Table 4.3.2 Encrypting a letter home

Input	Process	Output
Dear mum and dad, Thank you for encouraging us to enter the Young ICT Explorers competition this year. Our project titled "Vigener Extraordinaire" made it to the finals! I am writing to invite you both to accompany us to the University of Queensland where we will present our project to representatives from the Information Technology industry and the university. I hope you can come. Please let Aunt Edna know because she gave us the idea from that movie we all saw about the Imitation Game. Lots of love, Dawn and Bruce Secret key: exciting	<pre> graph TD Start([Start]) --> Click[Click on the padlock icon] Click --> Display[/Display dialog box/] Display --> EnterMsg[/Enter your message/] EnterMsg --> SelectOption[/Select option Encrypt/Decrypt/] SelectOption --> IsEncrypt{Is option Encrypt} IsEncrypt --> EnterKey1[/Enter secret key/] EnterKey1 --> PerformEnc[Perform encryption] PerformEnc --> OutputEnc[Output encrypted message] IsEncrypt --> IsDecrypt{Is option Decrypt} IsDecrypt --> EnterKey2[/Enter secret key/] EnterKey2 --> PerformDec[Perform decryption] PerformDec --> OutputDec[Output decrypted message] OutputEnc --> Stop([Stop]) OutputDec --> Stop </pre>	HBCZFCZGRAFIWBUGRHAWN - BXIKEWNZNMKICLBBKRQGM - PRESRPOBKGBMNWKMAYGLOXX - BVZMLPBAQFEIXTNWZCXSGGKM - BVZPBFDBORTIOGMBEGSOFQ - GIVXIJCLXQGZSQJMYQAGP - PKIFEEOXFPOMWVTZFVMRWH - HSQJBHPIISJRIGGHYXLVPX - CAOZBTABBLUJNWMXVFREK - FEAMEKABYQETCXIPGVMWHX - TOQRXKGZSOGXKMFKRQCBB - DRYJOQUMPRORCQZFIGOSKVM - VPAUPLIGBVQAWGTGTQZL - BWVBDRXWFGVBPBIVQCVI - AISJGXMNYIIGBTCAZCBFVT - SAUAYGKTCFKWEGOTDRAWQJM - BLRGJOQUMPNZQLXQXERGPIUI - PIOUYQVPXQZOXVQHVGTGQBN - WMABLPLXMWIJTEKFJKCPK

Algorithm for a polyalphabetic cipher

The IPO chart above (table 4.3.2) is at the top level (not broken down) and uses a flowchart to illustrate the algorithm. The same algorithm, written as pseudocode in some more detail, would be something like this:

//Function Vigenere

```
SET alphabetArray TO 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
GET message FROM user
GET secretKey FROM user

ASK user "Encrypt or decrypt?"

IF encrypt THEN
    SET mode TO encrypt
ELSE
    SET mode TO decrypt
ENDIF

CALL translateMessage(secretKey, message, mode)
```



Polyalphabetic encryption and decryption

You now need to break this down further into a translateMessage algorithm, based on what we learned above. Remember that in this program both the sender and the receiver use the same keyword for encryption and decryption. This is a simple method of shifting the letters in the message forwards or backwards depending on whether we are in encrypt or decrypt mode. Each time we encode or decode the next letter in the message, we use the next letter in the secretKey.

//Function TranslateMessage

```
SET translatedMessage TO [] //this will hold the encrypted or decrypted message
SET keyIndex TO 0 // this will be used to keep track of the letter position in the secret key
SET keylength TO length of secretKey

FOR each letter in message
    Find index1 of letter in alphabetArray
    Find index2 of secretKey[keyIndex] in alphabetArray
    IF mode = encrypt THEN
        SET newIndex TO index1 + index2
    ELSE
        SET newIndex TO index1 - index2
    ENDIF
    MOD newIndex by 26 //divide by letters in alphabet and take the remainder
    APPEND alphabet letter at alphabetArray[newindex] TO translatedMessage

    ADD 1 to keyIndex //move to next letter in the key
    MOD keyIndex by keylength //wrap around to start of key if needed
END FOR

OUTPUT translatedMessage
```

Algorithm desk checks

translateMessage desk check

A desk check or trace table is used to test algorithms or code to verify the accuracy of the algorithm or program logic. Each variable in the algorithm becomes a column in the test table.

ACTIVITY 4.3.2

Complete a desk check on the translateMessage algorithm using the following input data.

Test Case 1:

mode: encrypt

secretKey: cat

message: coding

Test Case 2:

mode: decrypt

secretKey: dog

message: gozd



STEP	INDEX1	INDEX2	KEYINDEX	MODE	TRANSLATEDMSG	MESSAGE	SECRETKEY	KEYLENGTH
1								
2								
3								
4								

The trace table can be extended to include as many rows as necessary to generate the required output.

Polyalphabetic cipher code example

The Python code below has been generated using the Vigenère algorithm above. The Rosettacode.org website includes additional Vigenère cipher code from 64 different programming languages.

```
alphabetArray = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'

def main():
    message=input("Enter your message: ")
    secretKey=input("Enter the secret key: ")
    mode=input("Enter 1 for encrypt or 2 for decrypt: ")

    if mode=='1':
        encrypt=True
    else:
        encrypt=False
    translateMessage(secretKey.upper(),message.upper(),encrypt)

def translateMessage(key, msg, encryptMode):
    ##### FUNCTION: translateMessage #####
    ### This function encrypts or decrypts a message using a      ###
    ### Vigenere cipher                                           ###
    ###                                                           ###
    ### INPUTS: key (string), msg (string), encryptMode (boolean) ###
    ### OUTPUTS: translatedMessage (string)                      ###
    ###                                                           ###
    #####

    translatedMessage=[]
    keyIndex=0
    keyLength=len(key)
    for letter in msg:
        index1=alphabetArray.find(letter)
        index2=alphabetArray.find(key[keyIndex])
        if encryptMode:
            newIndex=index1 + index2
        else:
            newIndex=index1 - index2
        newIndex %= 26
        translatedMessage.append(alphabetArray[newIndex])
        keyIndex+=1
        if keyIndex == keyLength:
            keyIndex=0
    print (" ".join(translatedMessage))
```



ACTIVITY 4.3.3

- 1 Using the algorithm provided on page 331, implement the Vigenère cipher in a programming language of your choice.
- 2 Use the test scenarios from the previous activity to test your code.
- 3 Using data from your testing scenarios, evaluate the accuracy of your encryption cipher.
- 4 Research variations to the Vigenère cipher.
- 5 Recommend and prioritise improvements for your Vigenère program.

One-time pad cipher

The most effective way of preventing any leaks of information and hiding your keyword from code breakers is to use randomness. In the previous cipher example, two people shared a keyword that increased the number of shifts. This, however, is still fixed. What if these same two people decided not to use a keyword but to use a matching series of randomly selected characters? This now increases the possibilities, as each letter shift is now independently random.

For example, imagine if the sender, instead of using a keyword, were to continuously select random letters, and share this list with the receiver as the key. It is important that the list be as long as the number of letters in the message to avoid repetition. The sender then uses the corresponding randomly selected letters to continually shift the original letter and create an encrypted set of data. This is sent to the receiver, who can then decrypt using the key in reverse.

This technique has two powerful security properties:

- no repetitive pattern – in other words, the shifts used to encrypt the message now do not repeat, as was the case in the keyword example
- no frequency differential – frequency is a count of the number of repeated letters occurring in a certain pattern. Having no repetition prevents leaks of information, making it harder for code breakers to decipher your keyword.

The one-time pad cipher, which emerged towards the end of the 19th century, is currently the strongest method of encryption. In order to understand it, you need to first understand the massive number of possible combinations that are involved.

The Caesar cipher used just one shift number, between 0 and 25, to shift every letter. For example, if you encrypted your name using the Caesar cipher you would have 26 possible solutions. That is a relatively small number of possibilities and makes the cipher easy to crack using what is known as a brute force search.

By comparison, in the following ‘one-time pad’ cipher example, each letter can be shifted in 26 different possibilities. Applying this to the name ‘MARY’, for example, there would be $26 \times 26 \times 26 \times 26 = 26^4$, or 456 976, possibilities. The longer the word, the more possibilities there are. Try listing out the first few possibilities, but don’t try them all – you won’t have enough time or paper.

One-time pad encryption and decryption: worked example

In the following table, the word ‘digital’ is shifted in relation the corresponding random character set ‘srutbdx’ to output an encrypted phrase.

Plaintext letters	d	i	g	i	t	a	l
Index of plaintext letters	3	8	6	8	19	0	11
Randomly generated corresponding letters	s	r	u	t	b	d	x
Index of random letters	18	17	20	19	1	3	23
Plaintext index + random index MOD 26	21	25	0	1	20	3	8
Encrypted letters	v	z	a	b	u	d	i

Conversely, the next table shows how the encrypted phrase is decrypted back to its original state.

Encrypted letters	v	z	a	b	u	d	i
Index of encrypted letters	21	25	0	1	20	3	8
Randomly generated corresponding letters	s	r	u	t	b	d	x
Index of random letters	18	17	20	19	1	3	23
Plaintext index – random index	3	8	–20	–18	19	0	–15
(If number is negative, add 26)			6	8			11
Encrypted letters	d	i	g	i	t	a	l

One-time pad encryption and decryption: generic algorithm

The development of randomly selected corresponding characters could be further unpacked to fully understand how encryption and decryption of a one-time pad cipher can be achieved. However, the following generic algorithm can be used to both encrypt and decrypt the series of characters.

```

INPUT message
INPUT random corresponding characters
FOR EACH letter in the message
  IF encryption
    Encrypted index = letter index + random
    letter index MOD 26
  ELSE (Decryption)
    Encrypted index = letter index – random letter index
    IF encrypted index < 0
      Encrypted index = encrypted index + 26
    ENDIF
  ENDIF
  ADD encrypted letter (using index) to encrypted message
ENDFOR
OUTPUT message
  
```

ACTIVITY 4.3.4

- 1 Based on the pseudocode at left and your knowledge of the code in the polyalphabetic cipher, generate code for the one-time pad cipher in your chosen platform.
- 2 Based on your understanding of the worked example for the one-time pad cipher, develop an algorithm and generate the code in your chosen platform for an extended one-time pad cipher that can handle any characters entered via a computer keyboard.

Quality of ciphers

One of the issues with the use of some encryption methods is the ease with which they may be broken, thus allowing unauthorised access to the information. Very simple algorithms such as the Caesar cipher can be broken. Other slightly more advanced algorithms are difficult to break by a human without a computer. When a computer becomes involved, a human can instruct it to use methods to detect and crack even a moderately encrypted message. The basic premise behind choosing a level or complexity of encryption is to make it too hard for attackers to crack. There should not be a risk that an attacker's computer could compute the possibilities in a reasonable and economical amount of time. For example, a banking transaction with an account balance of \$10 000 may be encrypted with a 512-bit key. To crack this, an attacker would have to use a disproportionate amount of computer power and time, making the task unviable. The computing cost alone might outweigh any potential value of gaining access to the data, and the time would be excessive.

Attackers use a range of known techniques to attempt to crack encrypted data. The most common is simple brute force – looking at each possible permutation for patterns of recognition. When the number of permutations to be processed is far beyond the limits of computation, the risk of detection becomes less. While there are a number of other factors involved, including predictability, markers and patterns, there are three main factors for determining a high-quality cipher:

- key length – as the key gets bigger, so does the complexity of trying to find all possible permutations
- the limitations of computing power
- the complexity of the cipher algorithm itself.

Moore's Law is a well-known prediction, which says that computer power roughly doubles every 18 months or so. While this might no longer be true one day, up until now it has been reasonably accurate. With computing power increasing, to make it unreasonable and uneconomical to crack encrypted data relies on increasing the complexity of the other two factors as well. This is why different standards of encryption have evolved over time and why, specifically, 128-bit key encryption standards have evolved to 256-bit, 512-bit and so on.

Modern ciphers

The ciphers employed by computer systems today still work on similar concepts to the classical examples explored earlier. A process of substitution from one state to another is at the centre of the cipher process, first to make data unrecognisable and then to return it to its original state. The processes still rely on a common auxiliary piece of information called a key, which is secretly shared and then used.

Modern ciphers attempt to be viable and safe solutions by considering the three factors mentioned previously. Emphasis has been placed not only on key size, but on the complexity of the algorithmic processes each employs to prohibit detection of elements, patterns, markers or possible permutations, and thus to protect against unauthorised decryption.

Modern ciphers can be generally categorised by two features: whether they are block or stream, and whether they are symmetric or asymmetric.

The difference between a block cipher and stream cipher is that in a block cipher, unencrypted (plain) binary text is broken into blocks or groupings, and processes are then performed to regenerate each block as encrypted text. Stream ciphers are technically like block ciphers except that the processes are performed one bit at a time. Stream ciphers are generally quicker than block ciphers, and are often deployed where speed is crucial and hardware resources are limited. While there is argument about which is the best, block ciphers are considered the most secure methods of encryption.

Whether a cipher is symmetric or asymmetric is to do with how keys are generated and how they are shared between parties. Asymmetric keys will be explored further in the next section. For now, only symmetric ciphers will be explored. As mentioned in section 4.1, there are a number of commonly used ciphers in the world today. Well-known algorithms include those in table 4.3.3.

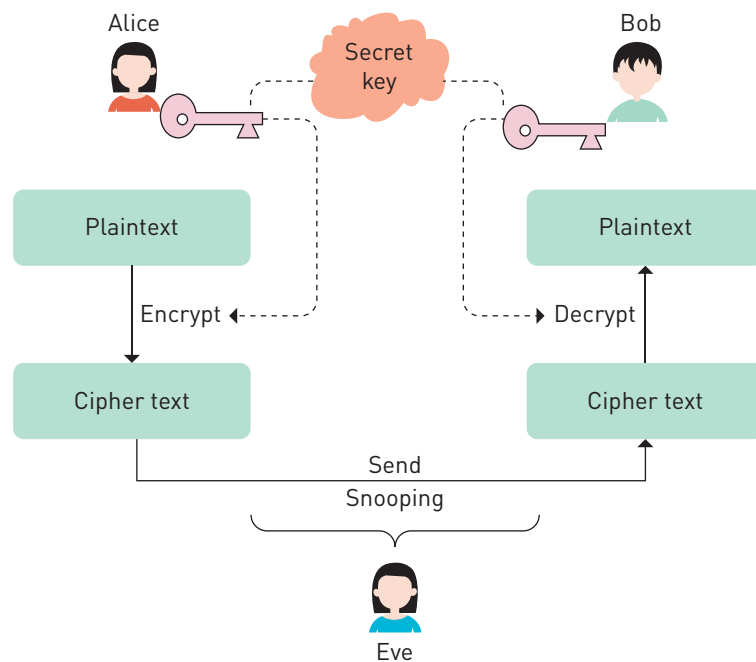


Figure 4.3.5 Symmetric encryption

Table 4.3.3 Properties of popular encryption algorithms

CIPHER ALGORITHM	BLOCK OR STREAM	SYMMETRIC OR ASYMMETRIC	INFORMATION
DES	Block	Symmetric	Developed in the 1970s and for many years one of the main algorithms used. It is considered a security risk today, but still offers an insight into the development of modern ciphers.
TripleDES	Block	Symmetric	An evolution of the original DES, it is still used in some situations, e.g. by banks to encrypt PINs at ATMs.
Blowfish and Twofish	Block	Symmetric	These are considered efficient open-source algorithms and are used in applications for encryption.
AES – Advanced Encryption Standard	Block	Symmetric	One of the most common and widely used algorithms. Considered a benchmark.
RSA	Block	Asymmetric	Widely used in network/Internet security. This will be explored in section 4.4.
RC4	Stream	Symmetric	Widely used in wireless protocols such as WEP and WPA.

Feistel cipher approach

Most block ciphers are based on a structure known as the Feistel cipher. This refers to a set of process patterns that are reused within the algorithm. While each cipher applies the ideas differently, they generally consist of multiple rounds of processing of the original data. Within each round a substitution (or swapping) event occurs, which is then followed by a permutation (shifting left or right – a bit like a Caesar alphabet shift) step.

Simplified DES cipher

Data Encryption Standard (DES) was initially one of the most commonly used symmetric block algorithms deployed for computer encryption. It was developed in the 1970s by IBM and, like many of its successors, is based on Feistel cipher concepts. While it is now considered insecure and is no longer commercially used, an exploration of it provides an insight into how modern cryptography is applied today.

The original DES algorithm has a number of features:

- It uses blocks of 64 bits. This size is fixed, meaning it has to pad them out if they are smaller.
- It has a 64-bit key, but only 56 bits are used by the algorithm.
- The key is used as part of transforming blocks of data.
- It has 16 rounds or passes at the data for encryption and decryption.

In exploring this algorithm, we will use a variation called 'Simplified DES'. The simplified version simply reduces the length of blocks, key and rounds. Other than that, it is very similar in approach.

The features of the Simplified DES algorithm are:

- It uses blocks of 10 bits.
- It uses a 10-bit key, of which only 8 bits are used.
- The key is still used as part of transforming blocks of data.
- It has only two rounds or passes at the data for encryption and decryption.

When using the algorithm, there are three separate processes:

- Generating the key
 - Key shared between sender and receiver
 - Subkeys created from the main key for use in encrypting/decrypting
- Encryption algorithm
 - Get data
 - Get subkeys
 - Break into blocks
 - Encrypt blocks
 - Output encrypted data
- Decryption algorithm
 - Get encrypted data
 - Get subkeys
 - Break into blocks
 - Decrypt blocks
 - Output data

To explore these three sections, we will use the Simplified DES bench-testing digital asset. This asset is available at the NelsonNet link.



Simplified DES
bench-testing
digital asset

Key generation algorithm

Before the S-DES cipher can be used, a key must be selected. This key must be a completely random 10-digit set of binary bits (1s and 0s). The key generation algorithm then uses this key to process additional smaller keys for use in the encryption and decryption process. Below is a sample version of the S-DES key generation algorithm.

INPUT a random 10 digit bit key

{COMPLETE Round 1}

Transform by changing the arrangement 12345678910 to 35274101986

Split key in half 35274 and 101986



STORE initial left and right splits

Left shift each by 1 in and curl back 52743 and 198610

JOIN and RESET POSITIONS as 1 2 3 4 5 6 7 8 9 10

TRANSFORM to eight digits 6, 3, 7, 4, 8, 5, 10, 9 (kill off 1 and 2)

STORE result as SUBKEY1

{COMPLETE Round 2}

LOAD initial left and right splits

LEFT SHIFT by 1 again, 43527, 610198

JOIN and RESET POSITIONS as 1 2 3 4 5 6 7 8 9 10

TRANSFORM to eight digits 6, 3, 7, 4, 8, 5, 10, 9 (kill off 1 and 2)

STORE result as SUBKEY2

The algorithm's goal is to take the initial random key that is either generated or supplied, and execute a series of rearrangements, shifts and reductions. The original 10-bit key is processed to make two different 8-bit subkeys, subkey1 and subkey2. These subkeys become the output for this algorithm and are then used during the encryption and decryption process.

If we explore elements of the supplied bench-test digital asset, it is possible to see what is occurring at each transformation. When a rearrangement occurs, we use the index positions of the elements to move them.

REARRANGEMENT	Positions before	1	2	3	4	5	6	7	8	9	10	Values before	1	1	0	0	1	1	0	1	1	0
(PERMUTATION)	Positions after	3	5	2	7	4	10	1	9	8	6	Values after	0	1	1	0	0	0	1	1	1	1

In this example, the left positions refer to the index of each value. For example, Position1 is Value 1, Position 4 is Value 0, Position 8 is Value 1. The before and after positions detail what exact process the algorithm must complete to rearrange the text elements.

RETRIEVE INITIAL SPLITS	Split 0 before	1	2	3	4	5						Values before	1	0	1	1	0					
(ALREADY SHIFTED ONCE)	Split 1 before						6	7	8	9	10							0	1	1	0	1
LEFT SHIFT BY 1 (WRAP AROUND)	Split 0 after	5	1	2	3	4						Values after	0	1	0	1	1					
	Split 1 after						10	6	7	8	9							1	0	1	1	0

In this example, a rotation of index positions dictates how the text elements must be transformed.

Encryption algorithm

The encryption algorithm completes its process by using both subkey1 and subkey2 to encrypt blocks of binary data. Each block represents 8 bits of binary data (for example, 10111001). One process that needs to occur before the encryption algorithm is the breaking of data into block size to process. Below is a sample version of the SDES encryption algorithm.

INPUT Plaintext (8 digit binary bits)

COMPLETE Initial Rearrangement

RESET Position Identifiers 1-8

SPLIT text 1-4 / 5-8

STORE Left Split

STORE Right Split (original - won't change)

EXPAND Right Split Only

XOR with Subkey1



RESET Position Identifiers 1-8
 SPLIT text 1-2 / 3-4
 COMPLETE Split 0 Matrix lookup – digits 1/2 as coordinates in table
 COMPLETE Split 1 Matrix lookup – digits 3/4 as coordinates in table
 COMBINE Output of Split 0 and 1
 RESET Position Identifiers 1-4
 COMPLETE Rearrangement
 STORE Right Split output
 XOR Right Split OUTPUT with STORED Left Split
 STORE result
 COMBINE result with STORED Right Split (original)
 SWITCH sub-blocks of combined result
 RESET Position Identifiers 1-8
 SPLIT text 1-4 / 5-8
 EXPAND Right Split Only
 XOR with Subkey2
 RESET Position Identifiers 1-4
 SPLIT text 1-2 / 3-4
 COMPLETE Split 0 Matrix lookup – digits 1/2 as coordinates in table
 COMPLETE Split 1 Matrix lookup – digits 3/4 as coordinates in table
 COMBINE Output of Split 0 and 1
 RESET Position Identifiers 1-4
 COMPLETE Rearrangement
 STORE Right Split output
 XOR Right Split OUTPUT with STORED Left Split
 STORE result
 COMBINE result with STORED Right Split (original)
 RESET Position Identifiers 1-8
 RECOMPLETE Initial Rearrangement
 OUTPUT Cipher text (8 digit binary bits) INPUT Plaintext (8 digit binary bits)
 COMPLETE Initial Rearrangement
 RESET Position Identifiers 1-8
 SPLIT text 1-4 / 5-8
 STORE Left Split
 STORE Right Split (original – won't change)
 EXPAND Right Split Only
 XOR with Subkey1
 RESET Position Identifiers 1-8
 SPLIT text 1-2 / 3-4
 COMPLETE Split 0 Matrix lookup – digits 1/2 as coordinates in table
 COMPLETE Split 1 Matrix lookup – digits 3/4 as coordinates in table
 COMBINE Output of Split 0 and 1
 RESET Position Identifiers 1-4
 COMPLETE Rearrangement
 STORE Right Split output
 XOR Right Split OUTPUT with STORED Left Split
 STORE result
 COMBINE result with STORED Right Split (original)
 SWITCH sub-blocks of combined
 RESET Position Identifiers 1-8

SPLIT text 1-4 / 5-8
 EXPAND Right Split Only
 XOR with Subkey2
 RESET Position Identifiers 1-4
 SPLIT text 1-2 / 3-4
 COMPLETE Split 0 Matrix lookup – digits 1/2 as coordinates in table
 COMPLETE Split 1 Matrix lookup – digits 3/4 as coordinates in table
 COMBINE Output of Split 0 and 1
 RESET Position Identifiers 1-4
 COMPLETE Rearrangement
 STORE Right Split output
 XOR Right Split OUTPUT with STORED Left Split
 STORE result
 COMBINE result with STORED Right Split (original)
 RESET Position Identifiers 1-8
 RECOMPLETE Initial Rearrangement
 OUTPUT Cipher text (8 digit binary bits)

The processes within the encryption algorithm are similar to those found in the full DES cipher.

Expand right split only	Positions before					5	6	7	8		Values before					1	1	0	1
	Positions after	8	5	6	7	6	7	8	5		Values after	1	1	1	0	1	0	1	1

In this example of expansion, a preset method is described using the value indexes of the right split of the binary text. Values in positions 5–8 are scrambled and duplicated to re-make another 8-digit text piece.

The other process new in this algorithm is XOR. XOR denotes a mathematical operation with binary digits.

XOR with subkey2	Each position value is compared. If same (e.g. 0 and 0 or 1 and 1) answer is 0, if different (e.g. 0 and 1 or 1 and 0) answer is 1.	Values before	1	1	1	0	1	0	1	1
		Subkey2	1	0	1	1	0	1	0	0
		Values after	0	1	0	1	1	1	1	1

As shown in the example from the S-DES bench-test digital asset, each digit in a position is compared to the corresponding digit in the second set. (for example, position 4 of value is compared to position 4 of the subkey). The combination of the two determines the course of action: A match (00 or 11) produces a 0, whereas a non-match (01 or 10) produces a 1.

The matrix lookup is another interesting feature.

Split	Split 0	1	2	3	4					0	1	0	1				
	Split 1					5	6	7	8					1	1	1	1
Split 0 matrix lookup	Bits 1 and 4 identify the row.	0,0	0,1	0,2	0,3					01	00	11	10				
	Bits 2 and 3 identify the column.	1,0	1,1	1,2	1,3					11	10	01	00				
	Each 2-digit binary number forms a decimal number between 0 and 3.	2,0	2,1	2,2	2,3					00	10	01	11				
		3,0	3,1	3,2	3,3					11	01	11	10				
Split 0 Output		01								Lookup cell:		1,2					
										Row:		01 (bits 1 and 4)					
										Col:		10 (bits 2 and 3)					

Split 1 matrix lookup

Bits 5 and 8
identify the row.
Bits 6 and 7
identify the
column.
Each 2-digit
binary number
forms a decimal
number between
0 and 3.

0,0	0,1	0,2	0,3
1,0	1,1	1,2	1,3
2,0	2,1	2,2	2,3
3,0	3,1	3,2	3,3

00	01	10	11
10	00	01	11
11	00	01	00
10	01	00	11

Split 1 output

11

Lookup cell:

[3,3]

Row: 11 (bits 5 and 8)

Col: 11 (bits 6 and 7)

Combine output of split 0 and 1

01 11

As shown in the bench-test example, the four binary digits of each split are used as coordinates to find replacement values from the matrix. These values are then used in later processes.

Decryption algorithm

As a requirement, the cipher is symmetrical in design. Therefore, the encryption and decryption algorithms must produce exactly the same output – except that in the case of the decryption algorithm this must be in reverse. The S-DES decryption algorithm mirrors the encryption algorithm except for when it uses the subkeys. In the decryption algorithm the subkeys are used in reverse. The remaining features are those already explored in the encryption algorithm. Below is a sample version of the S-DES decryption algorithm.

INPUT Cipher text (8 digit binary bits)

COMPLETE Initial Rearrangement

RESET Position Identifiers 1–8

SPLIT text 1–4 / 5–8

STORE Left Split

STORE Right Split (original – won't change)

EXPAND Right Split Only

XOR with Subkey2

RESET Position Identifiers 1–8

SPLIT text 1–2 / 3–4

COMPLETE Split 0 Matrix lookup – digits 1/2 as coordinates in table

COMPLETE Split 1 Matrix lookup – digits 3/4 as coordinates in table

COMBINE Output of Split 0 and 1

RESET Position Identifiers 1–4

COMPLETE Rearrangement

STORE Right Split output

XOR Right Split OUTPUT with STORED Left Split

STORE result

COMBINE result with STORED Right Split (original)

SWITCH sub-blocks of combined

RESET Position Identifiers 1–8

SPLIT text 1–4 / 5–8

EXPAND Right Split Only

XOR with Subkey1

RESET Position Identifiers 1–4

SPLIT text 1–2 / 3–4

COMPLETE Split 0 Matrix lookup – digits 1/2 as coordinates in table
 COMPLETE Split 1 Matrix lookup – digits 3/4 as coordinates in table
 COMBINE Output of Split 0 and 1
 RESET Position Identifiers 1–4
 COMPLETE Rearrangement
 STORE Right Split output
 XOR Right Split OUTPUT with STORED Left Split
 STORE result
 COMBINE result with STORED Right Split (original)
 RESET Position Identifiers 1–8
 RECOMPLETE Initial Rearrangement
 OUTPUT Plaintext (8 digit binary bits)

Sample code

Below are sample code components of Simplified DES.

```

private void GenerateSimpleDESKeys(out string key1, out string key2,
string inputKey)
{
    //Transform by changing the arrangement 12345678910 to 35274101986
    string keywork = inputKey.Substring(2, 1) + inputKey.Substring(4, 1) +
    inputKey.Substring(1, 1) + inputKey.Substring(6, 1) + inputKey.
    Substring(3, 1) + inputKey.Substring(9, 1) + inputKey.Substring(0, 1) +
    inputKey.Substring(8, 1) + inputKey.Substring(7, 1) + inputKey.
    Substring(5, 1);

    // Split key in half 35274 and 101986... STORE initial left and
    right splits
    string leftsplitkey1 = keywork.Substring(0, 5);
    string rightsplitkey1 = keywork.Substring(4, 5);

    //Left shift each by 1 in and curl back 52743 and 198610
    string leftsplitkey2 = leftsplitkey1.Substring(4, 1) + leftsplitkey1.
    Substring(0, 4);
    string rightsplitkey2 = rightsplitkey1.Substring(4, 1) +
    rightsplitkey1.Substring(0, 4);

    //JOIN and RESET POSITIONS as 1 2 3 4 5 6 7 8 9 10
    keywork = leftsplitkey2 + rightsplitkey2;

    //TRANSFORM to eight digits 6, 3, 7, 4, 8, 5, 10, 9(kill off 1 and 2)
    //STORE result as SUBKEY1
    key1 = keywork.Substring(5, 1) + keywork.Substring(2, 1) + keywork.
    Substring(6, 1) + keywork.Substring(3, 1) + keywork.Substring(7, 1) +
    keywork.Substring(4, 1) + keywork.Substring(9, 1) + keywork.
    Substring(8, 1);

    //LOAD left and right splits after shift
    //LEFT SHIFT by 1 again, 43527, 610198

    //Left shift each by 1 in and curl back 52743 and 198610
    leftsplitkey2 = leftsplitkey2.Substring(4, 1) + leftsplitkey2.
    Substring(0, 4);
  
```

```

rightsplitskey2 = rightsplitskey2.Substring(4, 1) + rightsplitskey2.
Substring(0, 4);

//JOIN and RESET POSITIONS as 1 2 3 4 5 6 7 8 9 10
keyword = leftsplitskey2 + rightsplitskey2;

//TRANSFORM to eight digits 6, 3, 7, 4, 8, 5, 10, 9(kill off 1 and 2)
//STORE result as SUBKEY2
key2 = keyword.Substring(5, 1) + keyword.Substring(2, 1) + keyword.
Substring(6, 1) + keyword.Substring(3, 1) + keyword.Substring(7,
1) + keyword.Substring(4, 1) + keyword.Substring(9, 1) + keyword.
Substring(8, 1);
}

```

In the example above, a function has been generated, demonstrating a sample round. Within this round, processes such as splitting, rearrangement, expansion and a matrix lookup have been described. As the cipher is symmetrical, the round can technically be used for both encryption and decryption.

```

private string matrixlookup0(string partkey)
{
    //COMPLETE Split 0 Matrix lookup - digits 1/2 as coordinates in table
    intRow = Convert.ToInt32(partkey.SubString(0,1)+partkey.SubString
(3,1), 2);
    intCol = Convert.ToInt32(partkey.SubString(1,1)+partkey.SubString
(2,1), 2);

    string[,] strMatrix = new string[4, 4] { {"01","00","11","10"},
                                                {"11","10","01","00"},
                                                {"00","10","01","11"},
                                                {"11","01","11","10"} };

    return strMatrix[intRow, intCol];
}

```

Symmetric vs. asymmetric encryption

Up until now, the ciphers that have been explored, developed, generated and evaluated have required one secret piece of information – a key. For these sorts of ciphers to be deployed into real use, both participants in the data exchange (the sender and the receiver) need to know the secret key before the data is exchanged. This concept of the shared key is commonly referred to as symmetric encryption. While symmetric encryption may have worked in Caesar's time, and does work when the key is predetermined and carefully shared within an organisation, it is simply not a viable way to keep data safe on open networks such as the Internet. This is because most exchanges need to occur quickly, without parties getting together to decide on a shared key. This is where the concept of asymmetric encryption comes into play.

Asymmetric encryption is a vital method of security on the Internet. Banks, governments and other organisations rely on asymmetric encryption to complete financial transactions safely and securely, to access and exchange sensitive information and to stop snooping of information by third parties.

Figure 4.3.7 details a classic example of the features of asymmetric encryption and how it is conducted between two parties. The most significant difference between symmetric and asymmetric encryption is the distinction between the public and private key.

In asymmetric cryptography (the creation of asymmetric ciphers), a system will develop two separate but linked keys – a public key, which is given to users who wish to send the system encrypted information (or their computers), and a private key, which is kept totally secret and is used to decrypt what is received from others. The theory behind this approach is that anyone with the public key can encrypt data, but the decryption can only be achieved by using the secret private key. Therefore, when a user sends through a

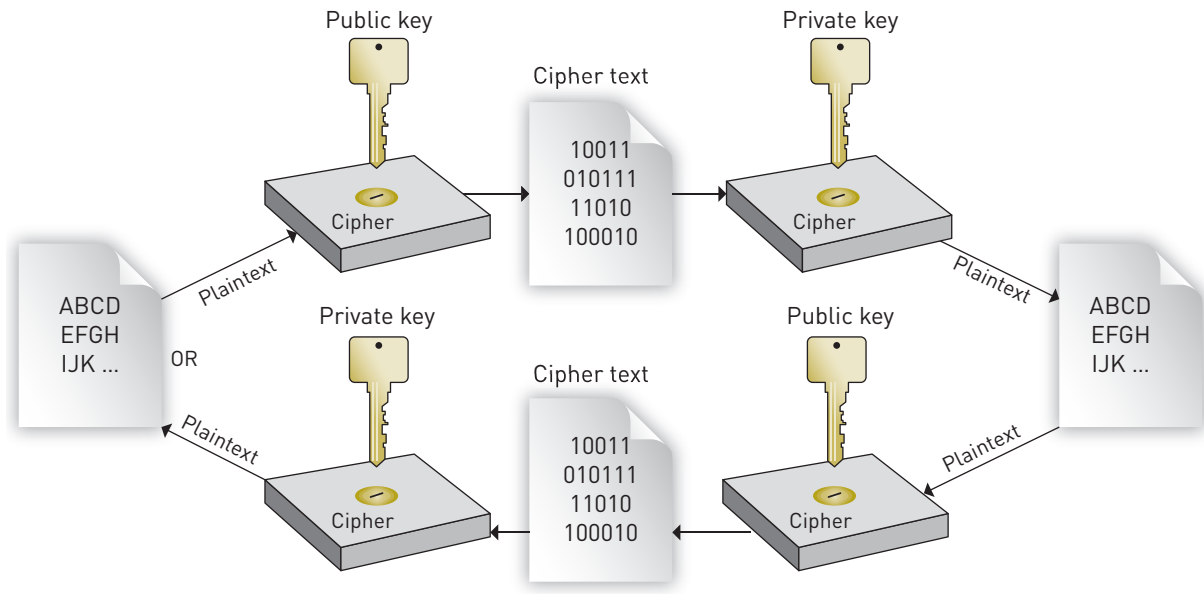


Figure 4.3.6 The process of asymmetric encryption and its use of public and private keys

request in an online banking application to transfer money, the bank is theoretically the only party that is able to understand and read the request. All other parties (snoopers, attackers) that potentially inspect the data along the public paths of the Internet are unable to read it or make any sense of it.

HTTPS

As mentioned in section 4.1, the S in HTTPS stands for 'secure'. HTTPS is a secure version of HTTP that uses public/private key cryptography between parties on the Internet. When a web page or data needs to be sent, the receiving web server will first send a unique set of encryption methods called an SSL certificate. With this, the web browser is able to access a public encryption key to begin the data exchange. This public key is then used in an initial introductory process called a 'handshake' to establish the secure session.

Once the handshake process is complete and verified, the data can be exchanged in a secure way for the session.

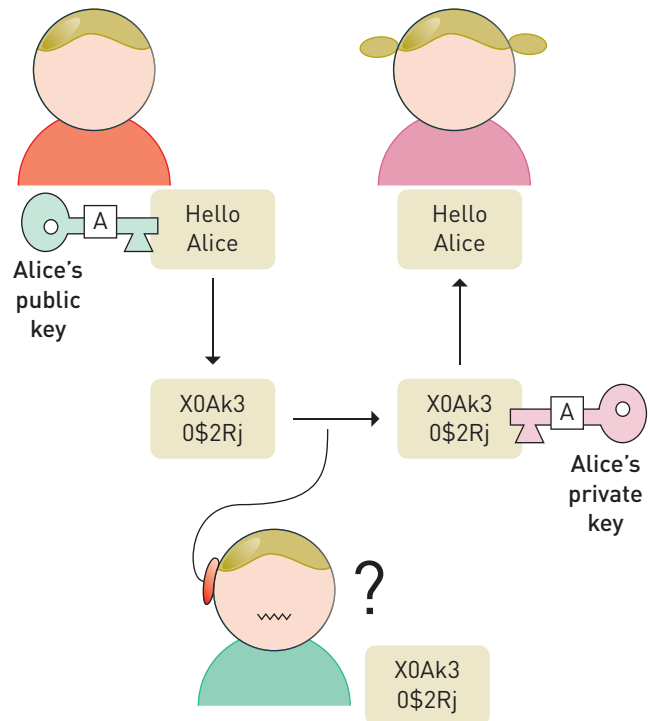


Figure 4.3.7 The message can only be understood by using the private key.

RSA

RSA encryption was one of the first widely used forms of asymmetric encryption algorithm. It was discovered by Ron Rivest, Adi Shamir and Leonard Adleman at the Massachusetts Institute of Technology (MIT) in the late 1970s. It works in a practical sense on the level of difficulty of completing factorisation

of the product. The mathematics behind RSA is quite advanced. We will explore a simplified variation of RSA and simulate an RSA-style process in the form of an algorithm and generated code segment. This still requires some neat mathematics, and may require some additional support and exploration to fully interpret.

Simplified RSA example mathematical process

Step 1: Creating a public and private key set

- 1 To begin, select two prime numbers. In this example, 5 (p) and 13 (q) will be chosen.

$$p = 5, q = 13$$

- 2 We must now complete the first calculation to create part of the public key (n). To achieve this, we simply multiply the two prime numbers (p and q) together.

$$\begin{aligned} n &= p \star q \\ &= 5 \star 13 \\ &= 65 \end{aligned}$$

- 3 We must now complete the initial part of the second calculation to create the other part of the public key. We need to calculate the value z . This is calculated via the following equation: $z = (p - 1) \star (q - 1)$.

$$\begin{aligned} z &= (p - 1) \star (q - 1) \\ &= (5 - 1) \star (13 - 1) \\ &= (4) \star (12) \\ &= 48 \end{aligned}$$

- 4 Now, using what we have discovered in z as a guide, pick another prime number that is not divisible within the z value (48). In this case, 11 has been chosen and is represented as k . (Note: 5 or 7 could have also been chosen.) So the complete public key ($n = 65$ and $k = 11$) has now been calculated.

$$\begin{aligned} 48 \text{ MOD } 3 &== 0? \text{ (NO)} \\ 48 \text{ MOD } 7 &== 0? \text{ (NO)} \\ 48 \text{ MOD } 11 &== 0? \text{ (NO)} \\ 48 \text{ MOD } 15 &== 0? \text{ (NO)} \\ 48 \text{ MOD } 17 &== 0? \text{ (NO)} \end{aligned}$$

{pick one}

- 5 The next step is to calculate the other key, the private key (j). To do this we need to discover j from this equation:

$$k \star j \text{ MOD } z = 1$$

We know the values of k and z , so we must determine a feasible value for j .

$$11 \star ? \text{ MOD } 48 = 1$$

For those who struggle with this maths, here is a brute force or trial-and-error approach.



Figure 4.3.8 HTTPS is a secure protocol for transferring data on the Internet

Refer to Section 4.1 to review an example of the handshake process.



J	J * 11
5	55
9	99
13	143
14	154
18	198
22	242
23	253
26	286
27	297
30	330
31	341
34	374
✓ 35	385
36	396

MULTIPLES OF 48	MULTIPLES OF 48 (+ 1)
48	49
96	97
144	145
192	193
240	241
288	289
336	337
384	385
432	433
480	481
528	529
576	577
624	625
672	673

So in this case, $j = 35$ would balance the result of this equation.

Step 2: Encrypting a message

Now, we have to encrypt something so that it can be transmitted. To do this we will add our value (V) to the equation to provide our encrypted result (E). To keep it simple, we will just use a random number as our message – in this case, 42.

$$\begin{aligned} \text{Encryption Equation: } V \wedge k \text{ MOD } n &= E \\ 42 \wedge 11 \text{ MOD } 65 &= E \\ 717368321110468608 \text{ MOD } 65 &= 48 \end{aligned}$$

So, our encrypted message (E) is now represented as 48.

Step 3: Decrypting the message

$$\begin{aligned} \text{Decryption Equation: } E \wedge j \text{ MOD } n &= V \\ 48 \wedge 35 \text{ MOD } 65 &= V \\ 6.9733796196349E+58 \text{ MOD } 65 &= 42 \end{aligned}$$

So, our received encrypted message (E) is decrypted using the private key ($j = 65$) to produce the original unscrambled value (V) as 42.



Simplified RSA: sample algorithm

The following are algorithms for the previous simplified RSA example.

Table 4.3.4 Calculating the public key and private keys

Input	Process	Output
Two random prime numbers - p (Prime1) and q (Prime2)	<pre> STORE Prime1 STORE Prime2 SET Publickey_Part1 = Prime1 * Prime2 SET Publickey_Part2 = (Prime1 - 1) * (Prime2 - 1) SET randomPRIMEFound = FALSE WHILE randomPRIMEFound == FALSE SET Prime3 = Random Number Crypto Function IF Publickey_Part2 MOD Prime3 <> 0 THEN RandomPRIMEFound = TRUE FOR each NUMBER between 1 and Publickey_Part2 - 1 IF Prime3 * NUMBER MOD Publickey_Part2 == 1 THEN SET Private_Key = NUMBER ENDIF END FOR END IF END WHILE </pre>	Public Key components (n (Publickey_Part1) and k (Publickey_Part2)), z (Prime3) and j (Private_key)

Table 4.3.5 Encrypting message using public key

Input	Process	Output
Public Key components And message (as a number)	<pre> GET MESSAGE ENCRYPTED MESSAGE = MESSAGE ^ Publickey_Part2 MOD Publickey_Part1 OUTPUT ENCRYPTED MESSAGE </pre>	Encrypted message (as a number)

Table 4.3.6 Decrypting message using private key

Input	Process	Output
Private Key components And encrypted message (as a number)	<pre> GET ENCRYPTED MESSAGE MESSAGE = ENCRYPTED MESSAGE ^ Private_Key MOD Publickey_Part1 OUTPUT MESSAGE </pre>	Decrypted message (as a number)

Encryption via code libraries

In most situations, using high-level encryption processes can be achieved easily with the use of existing code libraries within a nominated coding environment.

Below are some brief code examples using the Triple DES cipher. These examples show ways in which modern code libraries and cryptography features of language environments can be deployed to solve data security digital problems. It should be noted that these examples may not be executed without additional elements such as importing libraries and alignment of editions. For a full example of this code, see the examples on NelsonNet.

Triple DES symmetric encryption example

```
private void EncryptText(string plaintext, string randomkey, out byte[]
encryptedtext)
{
    System.Text.UTF8Encoding UTF8FormatStore = new System.Text.
UTF8Encoding();

    MD5CryptoServiceProvider HashProvider = new
MD5CryptoServiceProvider();
    byte[] TripleDESKey = HashProvider.ComputeHash(UTF8FormatStore.
GetBytes(randomkey));
    TripleDESCryptoServiceProvider TripleDESProcess = new
TripleDESCryptoServiceProvider
    {
        Key = TripleDESKey,
        Mode = CipherMode.ECB,
        Padding = PaddingMode.PKCS7
    };

    byte[] plaintext_converted = UTF8FormatStore.GetBytes(plaintext);
    ICryptoTransform TripleDESEncryptorProcess = TripleDESProcess.
CreateEncryptor();
    encryptedtext = TripleDESEncryptorProcess.
TransformFinalBlock(plaintext_converted, 0, plaintext_converted.Length);
}
```

Triple DES symmetric decryption example

```
private void DecryptText(byte[] encryptedtext, string randomkey, out string
plaintext)
{
    System.Text.UTF8Encoding UTF8FormatStore = new System.Text.
UTF8Encoding();

    byte[] Results;

    MD5CryptoServiceProvider HashProvider = new MD5CryptoServiceProvider();

    byte[] TripleDESKey = HashProvider.ComputeHash(UTF8FormatStore.
GetBytes(Passphrase));
    TripleDESCryptoServiceProvider TDESTripleDESProcess = new
TripleDESCryptoServiceProvider
    {
        Key = TripleDESKey,
        Mode = CipherMode.ECB,
        Padding = PaddingMode.PKCS7
    };

    ICryptoTransform Decryptor = TDESTripleDESProcess.CreateDecryptor();
    Results = Decryptor.TransformFinalBlock(encryptedtext, 0,
encryptedtext.Length);

    plaintext = UTF8FormatStore.GetString(Results);
}
```




TEST YOUR KNOWLEDGE

- 1 Brainstorm a list of situations where the security of data is important when communicating across the Internet.
- 2 Explain the difference between encryption and decryption.
- 3 In terms of digital data security, what is a cipher?
- 4 Draw a diagram to help explain how a Caesar cipher would help prevent a code-breaker accessing a message between two people.
- 5 Construct an infographic of the historical progression of data security between people, starting with the very first known cipher technique.
- 6 Explain why the concept of a key is central to all ciphers.
- 7 How are the two tables ASCII and UTF-8 significant in making ciphers more practical for use on computers?
- 8 Develop a list of all the ciphers mentioned in this unit in order from least secure to most secure. Award each cipher a security rating of between one and five stars based on your exploration of these ciphers.
- 9 In the list you created for question 8, indicate next to each cipher the technique used for its security – for example, single key shift.
- 10 In pairs use the polyalphabetic cipher to send each other a word – for example, your name. Take turns at being the sender and the receiver.
- 11 The polyalphabetic cipher algorithm in this section did not include any user proofing. Recommend and implement two changes to your Vigenère program that would improve the error handling of the program.
- 12 In terms of code breaking, what does an information leak mean?
- 13 Using your name as an example, explain how the one-time pad cipher is considered one of the strongest ciphers that involves a key.
- 14 List all control structures and data structures used in the one-time pad cipher algorithm.
- 15 Explain how the use of parameters in the Vigenère cipher made the translateMessage function more efficient and general purpose.
- 16 Explain the difference between a public and a private key.
- 17 Demonstrate how to calculate the number of possible solutions to a 128-bit key.
- 18 Describe the terms 'block', 'stream', 'symmetric' and 'asymmetric' as they apply to the categorisation of modern ciphers.
- 19 List the features of the original DES algorithm.
- 20 Explain the difference between the HTTP and HTTPS protocols.
- 21 Describe and summarise the RSA encryption / decryption process.



APPLY YOUR KNOWLEDGE

- 1 Explore the Gronsfield cipher. Use the Digital Solutions problem-solving process to develop an algorithm for this cipher.
- 2 Develop an algorithm and code for an XOR function that uses two 8-bit values.
- 3 Generate code for the Simplified RSA algorithm listed in this section using a chosen programming language.
- 4 Generate a coded Blowfish cipher encryption and decryption using appropriate code libraries.

4.4

EVALUATING THE IMPACTS OF DATA EXCHANGE AND OPEN DATA

In this section, you will evaluate the personal, social and economic impacts of sharing and exchanging data between systems via Internet protocols. You will use Structured Query Language (SQL) to simulate the creation of a de-identified data source (database tables) that can be accessed by an API for the purpose of exchange. You will use the Digital Solutions problem-solving process to explore how to de-identify sample data and to ensure that privacy principles are upheld.



Investigating impacts for storing and exchanging data

Data is a valuable commodity and is a source of revenue for many companies. Organisations compete to collect data from individuals about their product preferences, buying patterns and even political opinions. Some data collected by an organisation is retained by the organisation and not shared. This type of data can include confidential information about customers or staff and internal data about the business, which is used exclusively by the organisation and secured within a private organisational network. Sometimes, data collected by one organisation is also of value to another organisation. Before this data is shared, it may need to be edited to remove sensitive or confidential data items.

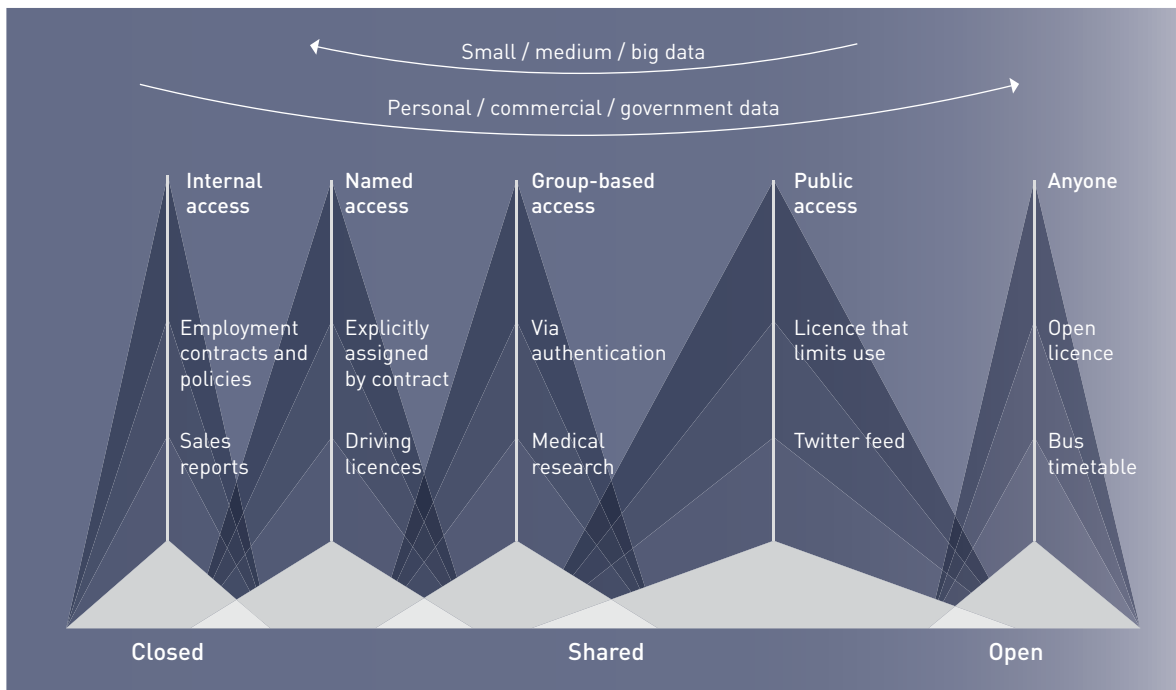


Figure 4.4.1 Data can be published with varying degrees of openness.

Access to the data will usually require an authorisation process, and it may be sold rather than given away. Purchasing access to reliable, quality data exchanged in a flexible, accessible format allows organisations to focus on using the data effectively within their organisation to increase revenue, reduce costs or improve client service without the hassle and expense of collecting the data themselves. Open data is accessible by anyone and is usually provided without charge or need for authentication. Open data exchanges encourage the use of data to benefit society without economic constraints. Not all data can be shared openly due to the need to protect personally identifiable data, but de-identified data (data that is separated from the individual it relates to) can be published freely to solve community problems that could not be addressed without access to open data.

The exchange and opening of datasets across business, government and society is expected to change how data is consumed, created and value added to. Open data movements and governments have accelerated their goals to produce levels of open data that can be used by communities, governments and businesses for both economic and social benefits. As with any new form of technology, there are negative consequences to these endeavours as well. The difficulty of maintaining privacy of the individual, security of sensitive data and the security of society as a whole mean that countermeasures are needed to aid the safe advancement of these innovations. Investigating social and economic impacts of open and exchanged data solutions will assist in identifying these impacts and how they can be celebrated, promoted and managed.

De-identification of organisational data

De-identification of data is the process of removing personal, sensitive or identifiable markers from a dataset. The data itself may require reworking or adjustment for this to occur.

As the amount of data within our institutions, businesses and governments increases rapidly, groups are looking for ways to provide opportunities for external bodies, companies, researchers and the general public to use data for different and more advanced purposes. The push for a more open use of data has amplified over the past decade and continues to gain momentum. The open data movement has spearheaded this campaign to highlight the social, cultural and economic benefits of using this data to generate new approaches and improve society through innovation.

ACTIVITY 4.4.1

Read the article 'Explainer: what is the open movement?' and consider the following questions.

- 1 What is the open movement?
- 2 What are the perceived advantages of the open movement?
- 3 How is licensing of the data handled? What are some of the common options used?
- 4 Research two technologies that have been designed to be open in nature, and describe the consequences of the decision to make them that way.



As much of the data housed by business and government can be highly sensitive in nature, to meet privacy expectations of the individual, open data at times requires adjustments to make it non-sensitive. One obvious way to do this is to step datasets through a process of de-identification, as previously explained in section 2.4.

PREPARING OPEN DATA

- De-identify the dataset for open consumption.
- Quality-assure the measures through consultations with privacy experts.
- Ensure that the Australian Privacy Principles are taken into account.
- Create a data dictionary to help people to use the datasets.

Privacy in Australia

In Australia, the *Privacy Act 1988* provides legal rights and regulations relating to how personal information is collected, handled and destroyed. The act describes personal information as ‘information or an opinion, whether true or not, and whether recorded in a material form or not, about an identified individual, or an individual who is reasonably identifiable’.

Individual information, in a general sense, is any element of information that is used to describe a person or can be linked to them. This includes data such as name, date of birth, unique identifiers such as driver’s licence number, passport number, tax file numbers, financial account details, signatures, address and contact information, medical records and taxation records. Data associated with this, such as sales information, logs, religious affiliations, memberships, political opinions and other personal traits are also deemed part of this if a person is identifiable, either directly or indirectly, from them.

Most Australians consider information privacy to be one of their basic rights. Most, however, find it difficult to describe what privacy is and what it means to them in detail. Privacy is best explained as a legal right to control how personal information is collected, used and handled. With the evolution of data-driven technologies, there are a number of pressures working against the notion of privacy. This section will explore some policies that assist individuals, organisations and governments to maintain this level of community expectation.

Australian Privacy Principles

The Australian Privacy Principles are a set of protocols that came into effect in 2014. They replaced the previous National Privacy Principles and Information Privacy Principles as a reference for individuals about how most organisations and governments in Australia are required to handle personal data. The principles are:

- 1 Open and transparent management of personal information
- 2 Anonymity and pseudonymity
- 3 Collection of solicited personal information
- 4 Dealing with unsolicited personal information
- 5 Notification of the collection of personal information
- 6 Use or disclosure of personal information
- 7 Direct marketing
- 8 Cross-border disclosure of personal information
- 9 Adoption, use or disclosure of government-related identifiers
- 10 Quality of personal information
- 11 Security of personal information
- 12 Access to personal information
- 13 Correction of personal information.

The principles are designed to provide advice about how to implement elements of the Privacy Act, including:

- disclosure of how organisations manage personal information (for example, publicly available Privacy Policy)
- the differences between solicited and unsolicited personal information and how this is managed
- how organisations can use and potentially disclose personal information
- the rights of individuals to access or correct their information
- responsibilities for maintaining security
- responsibilities in relation to maintaining levels of quality of the information
- disposal and/or de-identification.

Three of the Australian Privacy Principles are closely aligned with data exchange and open data.

Principle 1: Open and transparent management of personal information

This principle highlights and prescribes efforts that organisations must take to meet the requirements of the Privacy Act and the Australian Privacy Principles in general. It provides advice to organisations about how to clearly communicate what information they collect, how they hold and use it, and who they can potentially disclose it to. This communication should also include ways in which individuals can access, correct or raise concerns about their information. The main communication strategy the principle suggests is the use of an organisational privacy policy. This policy must be accessible and free to all individuals at any time. For example, when accessing systems or exploring an organisational website, a privacy policy is generally an accessible document that is highlighted or mentioned. Adopting services provided by companies such as Apple also directs users back to privacy disclosure statements that they must acknowledge.

Principle 6: Use or disclosure of personal information

This principle outlines how organisations may use personal information and in what circumstances they are allowed to disclose it. In a general sense, data must not be disclosed for any use other than its primary purpose unless consent is provided or other laws require it to be disclosed. In most circumstances, permission for third-party disclosures is listed among privacy policies that users are asked to agree to before they receive a service. In all other circumstances, information that is either de-identified or not considered personal or sensitive to an individual may be released. For example, if an organisation wanted to share elements of your personal data with a third-party research group, they would have to acknowledge this in their privacy policy or actively seek your consent.



4.4.2 Office of
the Australian
Information
Commissioner

Principle 11: Security of personal information

This principle outlines the requirement on organisations to take reasonable steps to protect an individual's personal information from misuse, interference, loss or unauthorised access or disclosure. Methods of security employed include technical and human processes such as encryption, access privileges and restrictions on how the information is accessed – for example, processes around how a customer can call an organisation and gain access to their information. Questions that substantiate a user's identity are among the methods used at present.

The principle also denotes that actions should be taken to destroy or de-identify data that the organisation no longer needs.

CASE STUDY

HERITAGE HEALTH PRIZE

De-identification of health data was used in the Heritage Health Prize (HHP), a competition launched by the Heritage Provider Network (HPN). HPN, a health maintenance organisation based in California, launched its Heritage Health Prize in April 2011. The objective of the competition is to construct a model to predict the number of days a patient will be hospitalised in the following year, by using the current and previous years' claims data. The winner, which is the team or individual with the most accurate model, receives a US\$3 million cash prize.

The competition requires that large datasets containing private health information of patients be made publicly available. In the United States there is no legislative requirement to obtain patient consent to disclose health information if the data is deemed de-identified under the *Health Insurance Portability and Accountability Act* (HIPAA). Therefore, all datasets to be used in the competition were de-identified and management of the risk of re-identification of datasets was addressed.

De-identifying techniques

- 1 Pseudonyms for member ID, provider ID, vendor etc. were used.
- 2 Top-coding was used. Top-coding is a process in which quantitative values that are considered uncommonly high are limited to an upper bound. This reduces the risk of individuals standing out.
- 3 Truncation of claims. The data involved a number of health claims made by patients. It is plausible that an individual could be identified if their demographic information was known, and it was known they had made an unusually large number of claims. Therefore, the number of claims per patient at the 95th percentile was truncated in order to reduce the risk of re-identification of 'extreme outliers'.
- 4 Removal of high-risk patients and claims. Patients who were considered to be high risk were removed from the dataset to avoid the chance that their disease, condition, or procedure could be inferred from patterns in the data.
- 5 Suppression of provider, vendor, and PCP identifiers. It was possible that providers could have patterns of treatment that make them stand out. For example, a provider could be the only one with a particular specialty in a specific place of service who performed procedures on patients with a particular diagnosis. For these records the provider ID was suppressed.

Re-identification risk assessment

A re-identification risk assessment of HHP datasets found the risk exposure was considered very low under the HIPAA Safe Harbor de-identification standard. The assessment also concluded that the HHP provides an example of, and methodology for, achieving open data principles for longitudinal health data, using good de-identification practices. De-identifying data allowed for the disclosure of data regarding individuals while simultaneously protecting those individuals from the risk of re-identification.

Source: © State of New South Wales (Office of Finance and Services) 2015, CC BY 4.0



Figure 4.4.2 De-identification of health data

iStock/Getty Images Plus/scanrail



4.4.3 The heritage health prize



4.4.4 Open data institute

Evaluating impacts of open data

Opening up data can potentially have massive benefits to communities, governments and economies.

From a government perspective, opening up public-sector data can assist in making government function more efficiently, and can potentially generate revenue, promote better decision-making and establish a platform for further innovation.

The initial investment in adjusting, preparing and de-identifying data is the main additional cost of this process.

Governments have also recognised that open and accessible data can help business, individuals and not-for-profit organisations within their jurisdictions, by reducing government-established economic constraints, promoting a better economic function and overall economic growth.

Studies have emerged to support these claims. Recent estimates of the potential value of open government data in Australia range into the tens of billions of dollars per annum. In a Lateral Economics report commissioned by the Omidyar Network, authors Nicholas Gruen, John Houghton and Richard Tooth suggested that a number of economic areas could make significant gains from the openness of data. It was suggested that the use of open data could lead to:

- more efficient practices in managing economic cycles, to the value of \$3.6 million per year
- a potential reduction in corruptive practices by \$1.5 billion per year
- maximising the output of existing infrastructure and supporting efficient development of new infrastructure. The combined net benefit of this to the Australian economy could be as much as \$7.1 billion on an annual basis.

GROUP ACTIVITY 4.4.1

In groups of four, read the Lateral Economics report into the benefits of open data in the Australian economy. In your group, explore key ideas of the case study and summarise key elements of the paper. In subgroups of two, select one economic area (such as employment, education, finance or law) and develop ideas about the ways in which open data could assist in economic advancement.

From an end user's perspective, open data can make a range of tasks and functions much easier. This can be achieved using quite rudimentary data, which might be sitting within the confines of a government department and from their perspective may not be properly valued or realised. Releasing this data can have a positive effect on people's daily lives and can improve the functioning of society.

CASE STUDY

DIGITAL TV ANTENNAS ANDROID APP

The Digital TV Antennas app is designed to support individuals adjusting their television antennas to gain the best signal strength. It works by mashing together open datasets supplied by the Australian Government in conjunction with the GPS signalling within the Android device.

Other countries such as the United States, Canada, the United Kingdom, New Zealand, Austria, Argentina, Brazil, Portugal, Colombia and Uruguay have also supplied datasets.

The working together of these two data sources provides an innovative, easy solution that a person can use while directly adjusting their antenna on rooftops, in caravans and campers or with portable television devices.



Figure 4.4.3 Digital TV Antennas Android app

4.4.5 How open data can help achieve the G20 Growth target

4.4.6 Licensed broadcasting transmitter data

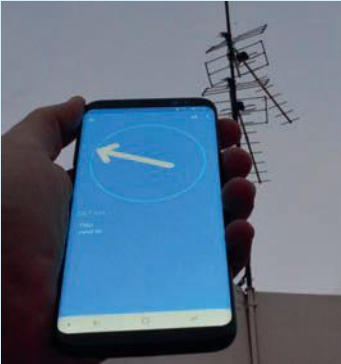


Figure 4.4.4 Open datasets working together with GPS signalling provide an innovative, easy solution in various contexts.

ACTIVITY 4.4.2

Read the article 'The future will be built on open data – here's why'. After reading this article, your task is to evaluate an open data initiative that has occurred in the last two years. In evaluating this initiative, consider the following issues:

- 1 How has the initiative improved on previous function?
- 2 What data had to be made accessible?
- 3 How was the data made open or shared? What has changed?
- 4 What other data sources or technologies were used in conjunction with this initiative?
- 5 Were there any actual or potential negative consequences of the initiative?



4.4.7 The future will be built on open data - here's why

Security of shared data exchange

As previously noted, it is not practical to make all data open. However, even datasets that still contain sensitive information might be shared between government departments and private organisations. This type of data sharing is only permissible when it is highlighted within an organisation's privacy policy or is mandated by law. Examples include sharing of:

- medical records between hospitals and researchers
- electoral rolls between government departments
- classified defence infrastructure arrangements between governments and contractors.

The issues related to the sharing of sensitive data are nothing new, but the push for this data to be exchanged rapidly across networks introduces additional risks. As previously explored, data must be secured by encryption and access privileges to stop unauthorised access. Data encryption techniques, along with secure transmission techniques such as HTTPS and underlying TLS security protocols, can significantly reduce risks. Ultimately, these security requirements depend on different organisations and users having secure computer hardware and practices. Analysts reiterate that end-point security is paramount to keeping data secure.



Figure 4.4.5 The rapid exchange of data across networks continues to make privacy and security essential.

ACTIVITY 4.4.3

Research the Australian Privacy Principles guidelines to find out what organisations are required to do, and what it is recommended they do, when they share sensitive data with third parties.



4.4.8 Australian privacy principles

ACTIVITY 4.4.4

Consider the response by AusCERT's manager in relation to Australia's eHealth dilemma.

- 1 What are some of the positive implications of sharing an individual's health records?
- 2 What are some of the negative implications of sharing an individual's health records?
- 3 What are the potential risks?
- 4 Who should be in control of the data? Why?
- 5 What strategies could be employed to improve the security of data across each part of the eHealth record system (individuals, doctors, hospitals, clinics and government agencies)?
- 6 If authorised access to an individual's health record was publicly shared, how could it affect their lives? Research this question to produce a considered response.



4.4.9 E-health record will be hacked, says Auscert

CASE STUDY

CLEVER WEB APPLICATION CONTEXT

Clever is an integrated student and teacher authentication and data sharing service available to schools and school sectors all over the world. While it is primarily focused on supporting schools within the United States of America, international schools, including some in Australia, are now using this service. The advantage of using Clever is that it can seamlessly connect student enrolment and timetable data between school systems and a vast number of cloud-based application services, including learning management systems, learning tools such as Mathletics and Khan Academy. It removes the need for teachers and students to self-create accounts and also centralises username and password authentications. It even makes secure login processes easier for younger students who may not know how to type or spell, by using unique QR codes and local device cameras.

When students log in, they can then select which cloud-based service they wish to use and those systems will know who they are and give them access.

The Clever service works by first collecting student data from schools on a daily basis. This information is sent by schools from their central student information systems to Clever via secure FTP or via secured API access using JSON data transfer techniques. The process occurs daily and refreshes data from the school. The data contains very detailed information about students, including names, identifiers, email addresses, date of birth, year groups, classes and teachers, as well as parent details.

A JSON representation of a student record is shown below:

```
{
  "data": {
    "id": "", //ObjectID: Globally unique and stable ID for student
    "district": "", // ObjectID: unique ID for student's district
    "school": "", //ObjectID: unique ID for student's primary school
    "schools": [ //List of ObjectIDs: schools student is associated with
      "",
      ""
    ],
    "created": "", //Timestamp: Resource initialisation date
    "last_modified": "", // Timestamp: Last time resource was modified
    "sis_id": "", //String: Internal student identifier from SIS
  }
}
```

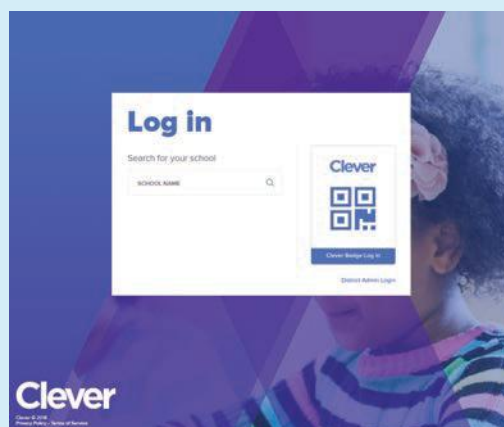


Figure 4.4.6 The Clever website bridges the gap between school systems and cloud-based application services.

```

"name": {
  "last": "", //String: Last name provided by district
  "middle": "", //String: middle name provided by district
  "first": "" //String: First name provided by district
},
"dob": "", //String: Student birthdate
"email": "", //String: Email address provided by district
"student_number": "", //String: Student number provided by district
"state_id": "", //String: State student identifier
"credentials": {
  "district_username": "" //String: District-specified student username
},
"gender": "", //String: Student gender
"race": "", //String: Student race
"hispanic_ethnicity": "", //Boolean: Student's hispanic/latino
ethnicity
"location": {
  "state": "", //String: Student state
  "zip": "", //String: Student zip
  "address": "", //String: Student street address
  "city": "", //String: Student city
  "lat": "", //String: Student address latitude (Deprecated)
  "lon": "" //String: Student address longitude (Deprecated)
},
"graduation_year": "", //String: Graduation year provided by district
"grade": "", //String: Student grade
},
"uri": "/v2.0/students/id"
}

```

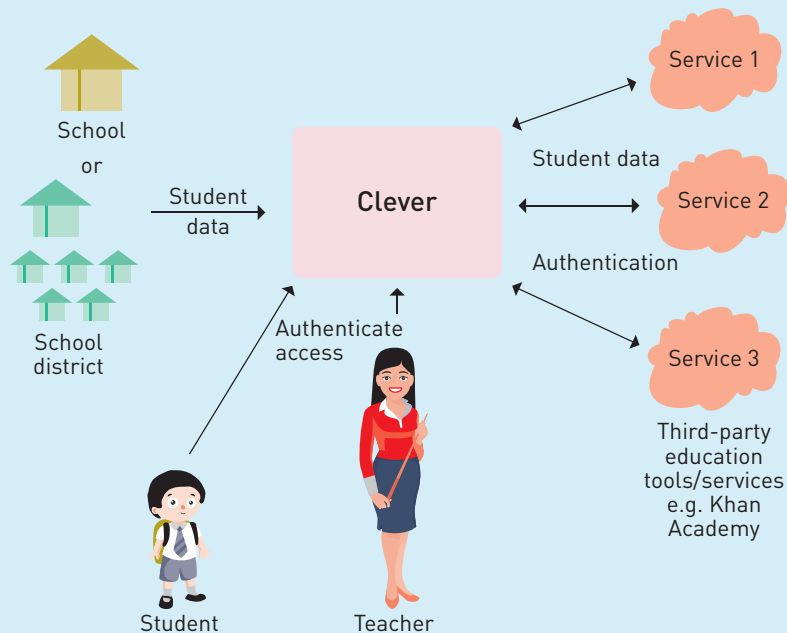


Figure 4.4.7 An illustration of how data is transferred between schools and cloud-based applications

Schools, students and teachers have found great value in the service. As a number of third-party services have access to sensitive information, the responsibility for security and privacy of the data is shared among all parties.

GROUP ACTIVITY 4.4.2

In groups of two, evaluate the Clever case study and complete the following tasks.

- 1 What are the advantages of sharing data in this case?
- 2 What barriers does this service overcome?
- 3 Who is ultimately responsible for the student data? Why?
- 4 In an Australian context, can a school successfully use this service and ensure compliance with the Privacy Act and associated principles?
- 5 What actions should an Australian school or district take to ensure compliance?
- 6 What risks are there in relation to sensitive data?
- 7 Are there any options available to schools/districts to modify their data before it is sent to Clever? Recommend three possible suggestions.

Generating de-identified and aggregated data

In many circumstances, organisations and governments are required to follow a process to adjust data for open consumption. In this section, a simulation of such a process will occur, giving opportunities to demonstrate more advanced features of Structured Query Language. This will build on knowledge and skills developed during Units 2, 3 and 4.

A fictitious scenario will be examined to follow de-identification strategies to produce a simplistic dataset for open consumption. Simulated tasks relating to relational data manipulation will occur.

Scenario: Fraser Coast Regional Council animal registry and complaints data

Imagine that the Fraser Coast Regional Council (FCRC) open data steering committee has begun examining the various datasets contained within the local government authority that it can contribute to its open data strategy to promote local innovation within the community.

The first open dataset that the steering committee has chosen to step through the open data process is its animal registry and complaints data.

The dataset for this type of data has already been archived by other local government jurisdictions and will be used as an initial test case to complete the process.

The committee has begun exploring the process it must undertake in order to achieve its goal of an open dataset. The council has already purchased an off-the-shelf API product that can pass data through a data exchange process based on JSON. The internal setup of the API is menu-driven. The administrator must simply attach a database as a data source and write SQL statements to harvest the data. The external API calls are then linked to the SQL statements and the fields are listed. The council's technical staff have already confirmed that a data dictionary is needed for this process.

During their first meeting relating to the animal registry and complaints data project, participants drafted the following mind map.

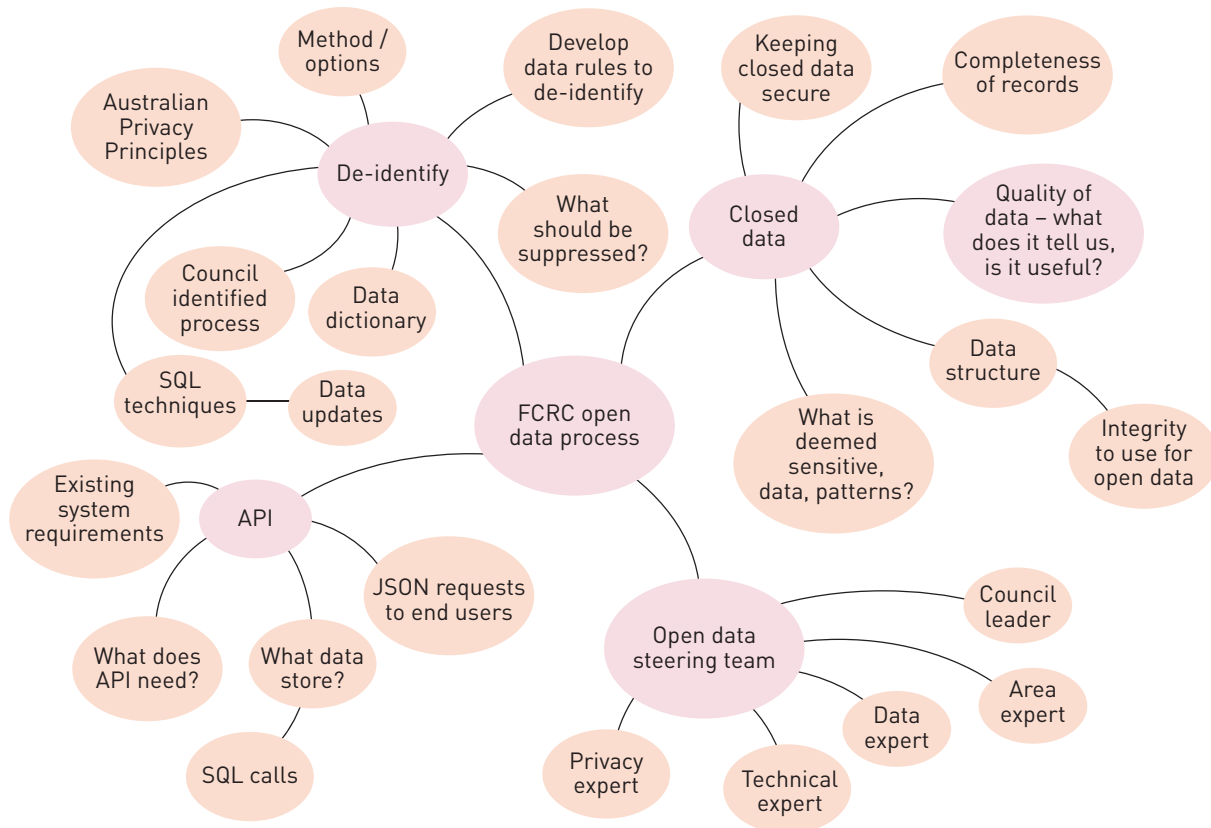


Figure 4.4.8 Mind map exploring the Fraser Coast open data problem

Council open data process

The committee has adopted a summarised process similar to those outlined by the Australian Government's Office of the Australian Information Commissioner for destroying or de-identifying personal information.

OPEN DATA PROCESS

- De-identify the dataset for open consumption.
- Quality-assure the measures through consultations with privacy experts.
- Ensure that the Australian Privacy Principles are taken into account.
- Create a data dictionary to help people to use the datasets.

These steps will be used to ensure that the data released meets community expectations of security and privacy while providing opportunities for innovative community use.

The scenario data

Within the council's information systems is a relational database instance that stores this information. The following relational schema (figure 4.4.9) outlines the basis of this system.

An investigation of the raw data has found that it appears to be complete. This is partially due to the mandatory requirements of fields in most of the tables and the rigidity of the database design to normalise animals, breeds, owners, complaint records and the type of complaints. Some of the details and notes listed within the system have been noted to be incomplete or lacking the necessary detail. This was particularly noticeable in

the early implementation stages of the information system. As the notes were found to be sensitive in nature or could have provided ways to identify individuals, they would most likely be suppressed from the open dataset and therefore do not require efforts to adjust or clean during such a process.

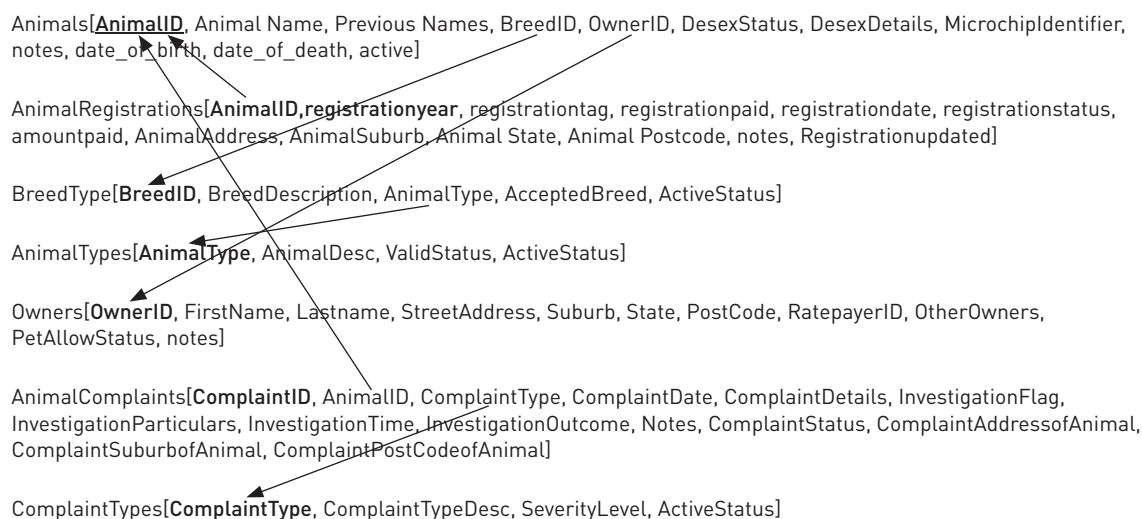


Figure 4.4.9 Relational schema for animal registry and complaints data

What must be de-identified

The steering committee has reviewed the raw dataset contained within the database structure. It has reviewed values and looked for personal identifiers, as outlined in resources provided by the Office of the Australian Information Commissioner. These describe the removal of personal identifiers such as name, address and date of birth. As well as this, the removal or modification of other ‘quasi-identifiers’ (for example, internal identifiers that may be linked within the organisation) that may possibly assist in identifying an individual should also be analysed and actioned. Other methods such as combining data elements from specific values into groups (for example, specific ages into age ranges) or scrambling pseudo-identifiers is also possible depending on the context of the open data being produced.

When modification options such as those listed are exhausted, it is then recommended that ‘synthetic data’ or ‘dummy data’ be generated from the original live data, or that aspects of the dataset are simply suppressed and not released.

In this situation, the steering committee has taken an approach to de-identify the data through:

- removal of personal identifiers (owner, animal, permits, dates of ownership, births)
- suppression of specific location data (number and street)
- suppression of complaint particulars (complaint notes or actions)
- suppression of specific dates. Partial data – month and year for complaints – will be allowed for grouping. Registration data will be suppressed to year grouping only
- release of categorised complaint data (complaint type only) specific to grouped locations (suburb only)
- release of certain complaint types only. There is a practice within council of updating the complaint type to a media-aware value. This flagged data will be suppressed
- release of categorised animal data (animal type) specific to grouped locations (suburb only)
- suppression of breeds to avoid unintended identification.

Advanced SQL techniques to de-identify and create open datasets

To complete the identification process, advanced Structured Query Language manipulations will be scripted. Through the use of table joins, grouping and subqueries, data will be manipulated, sanitised and extracted for use in the council's open data API.

Table joins

As previously explored in Unit 2, relational databases separate data across different tables to significantly reduce redundancy and improve data integrity and reliability. However, in most real-life uses of data, richer data must be brought together from different tables to give end users complete meaning. Table joins in SQL allow for this functionality.

An example using the council animal data might be to give richer meaning to a list of animals by providing their associated breed.

```
SELECT animalname, breedException
FROM council_animals.animals
INNER JOIN council_animals.breedtypes ON animals.breedid = breedtypes.breedid
```

ANIMALNAME	BREEDDESCRIPTION
Waggles	Border Collie
Apollo	Maltese
George	Labrador
Muppy	Labrador
Mildred	Golden Retriever
Digger	Silky Terrier

As shown in the example above, the animal name and the descriptive breedid have now been collated together for user consumption. In addition to this, the user may also wish to have a description of the animal type. To solve this, the join can be extended to involve multiple tables.

```
SELECT animalname, breedException, animaldesc
FROM council_animals.animals INNER JOIN council_animals.breedtypes
ON animals.breedid = breedtypes.breedid
INNER JOIN council_animals.animaltypes
ON breedtypes.animaltype = animaltypes.animaltype
```

ANIMALNAME	BREEDDESCRIPTION	ANIMALDESC
Digger	Silky Terrier	Dog
Mildred	Golden Retriever	Dog
Muppy	Labrador	Dog
George	Labrador	Dog
Apollo	Maltese	Dog
Waggles	Border Collie	Dog

Grouping

Grouping data into sets is a useful feature to assist in summarising data. The GROUP BY clause of SQL allows programmers and database administrators to generate powerful SQL statements for use.

A simple example of GROUP BY using the council animal data could look like this:

```
SELECT breedid, animalsuburb
FROM council_animals.animals INNER JOIN council_animals.animal_registrations
    ON animals.animalid = animal_registrations.animalid
GROUP BY breedid, animalsuburb
```

BREEDID	ANIMALSUBURB
2	Maryborough
3	Tiaro
1	Urangan
3	Burrum Heads
15	Pialba
4	Tiaro

As explored in Unit 2, The GROUP BY clause is often used in conjunction with aggregate functions. The previous example could also be extended to show the quantities of breeds in a given suburb.

```
SELECT COUNT(animals.animalid) AS number_of_animals, breedid, animalsuburb
FROM council_animals.animals INNER JOIN council_animals.animal_registrations
    ON animals.animalid = animal_registrations.animalid
WHERE registrationyear = 2018
GROUP BY breedid, animalsuburb
```

NUMBER_OF_ANIMALS	BREEDID	ANIMALSUBURB
1	1	Urangan
1	2	Maryborough
1	3	Burrum Heads
1	3	Tiaro
1	4	Tiaro
1	15	Pialba

The result of this SQL statement reveals a summary of the number of which breeds (shown as their id) exist in listed suburbs. In the case of the council animal database, a richer output can be achieved by also combining a table join.

```
SELECT COUNT(animals.animalid) AS number_of_animals, breeddescription,
    animalsuburb
FROM council_animals.animals INNER JOIN council_animals.animal_registrations
    ON animals.animalid = animal_registrations.animalid
    INNER JOIN breedtypes ON animals.breedid = breedtypes.breedid
WHERE registrationyear = 2018
GROUP BY breeddescription, animalsuburb
```

NUMBER_OF_ANIMALS	BREEDDESCRIPTION	ANIMALSUBURB
1	Border Collie	Urangan
1	Golden Retriever	Tiaro
1	Labrador	Burrum Heads
1	Labrador	Tiaro
1	Maltese	Maryborough
1	Silky Terrier	Pialba

To fully extend this example to include the animal description the following additions could be made.

```

SELECT COUNT(animals.animalid) AS number_of_animals, animaldesc,
       breeddescription, animalsuburb
FROM animals INNER JOIN animal_registrations
      ON animals.animalid = animal_registrations.animalid
      INNER JOIN breedtypes ON animals.breedid = breedtypes.breedid
      INNER JOIN animaltypes ON breedtypes.animaltype = animaltypes.animaltype
WHERE registrationyear = 2018
GROUP BY animaldesc, breeddescription, animalsuburb

```

NUMBER_OF_ANIMALS	ANIMALDESC	BREEDDESCRIPTION	ANIMALSUBURB
1	Dog	Border Collie	Urangan
1	Dog	Golden Retriever	Tiaro
1	Dog	Labrador	Burrum Heads
1	Dog	Labrador	Tiaro
1	Dog	Maltese	Maryborough
1	Dog	Silky Terrier	Pialba

When you explore the SQL results table, you can see that the grouping has been completed at multiple levels, firstly by suburb values, then by the animal description and finally by the breed description. The result of the COUNT() total is a reflection of these groupings.

GROUP BY ... HAVING

As explored in Unit 2, an additional feature of the GROUP BY clause is the ability to add conditions against grouped data. The HAVING clause can be used in conjunction with the GROUP BY clause. The HAVING clause is different from the WHERE clause as it is attempting to apply conditions to group results.

A simple example of a GROUP BY with a HAVING clause may look like this:

```

SELECT COUNT(animalid), breeddescription
FROM breedtypes INNER JOIN animals ON breedtypes.breedid = animals.breedid
GROUP BY breeddescription
HAVING COUNT(animals.breedid) > 1

```

COUNT	BREEDDESCRIPTION
2	Labrador

While the HAVING clause may be beyond the scope of the council animal de-identification process, an end-user example to filter out large numbers of animal breeds in suburbs may look like this:

```

SELECT COUNT(animals.animalid) AS number_of_animals, animalsuburb, animaldesc,
       breeddescription
FROM animals INNER JOIN breedtypes ON animals.breedid = breedtypes.breedid
      INNER JOIN animaltypes ON breedtypes.animaltype = animaltypes.animaltype
      INNER JOIN animal_registrations
      ON animals.animalid = animal_registrations.animalid
WHERE registrationyear = 2018
GROUP BY animalsuburb, animaldesc, breeddescription
HAVING COUNT(animals.animalid) = 10

```

In the above example, the previous grouped data has now been reduced to show only those breed numbers that are greater than 10 in a given suburb.

Subqueries

As previously explored in Unit 2, subqueries rely on different operands to connect the outer query and its subquery. Common operands include IN, ALL, ANY and =. The = operand can only be successfully used when there can only ever be a single result for the subquery. The IN operand is the most versatile and

commonly used operand for subqueries. It can be successfully used when there are anywhere between zero and N (0..N) possible results for the subquery. The ALL and ANY operands are most useful when linked with a comparison operator such as >=, >, < or <=.

For a detailed review of subqueries, see Unit 2



A subquery that could be used in the de-identification process might be constructed as follows:

```
SELECT COUNT(complaintID)
FROM animalcomplaints
WHERE complainttype IN (SELECT complainttype FROM complainttypes
                        WHERE severitylevel <> 3)
```

A more sophisticated version of data output that would result in suburb-specific animal type filtered counts may be generated using the following SQL statement:

```
SELECT COUNT(complaintid) AS number_of_complaints, complainttypedesc,
       complaintsuburbofanimal
FROM complainttypes INNER JOIN animalcomplaints
   ON complainttypes.complainttype = animalcomplaints.complainttype
WHERE animalcomplaints.complainttype IN (SELECT complaintType FROM ComplaintTypes
                                         WHERE SeverityLevel <> 3)
GROUP BY complaintsuburbofanimal, complainttypedesc
}
```

Generating the de-identified data

The steering committee has determined that two sets of data can be published for open use by the community. The two sets include:

- **Dataset 1**
Registered animal numbers across suburbs per year
- **Dataset 2**
A year and month summary of complaints against registered animals in a given suburb. This data will also include a breakdown of the non-extreme complaint types.

The following SQL statement has been produced to develop registered animal numbers across suburbs per year data for release:

```
SELECT COUNT(animals.animalid) AS numberofanimals, registrationyear,
       animalsuburb, animaldesc
FROM animals INNER JOIN animal_registrations
   ON animals.animalid = animal_registrations.animalid
   INNER JOIN breedtypes ON animals.breedid = breedtypes.breedid
   INNER JOIN animaltypes ON breedtypes.animaltype = animaltypes.animaltype
GROUP BY registrationyear, animalsuburb, animaldesc
ORDER by registrationyear, animalsuburb, animaldesc
```

NUMBEROFANIMALS	REGISTRATIONYEAR	ANIMALSUBURB	ANIMALDESC
1	2015	Maryborough	Dog
1	2016	Maryborough	Dog
1	2017	Maryborough	Dog
1	2018	Burrum Heads	Dog
1	2018	Maryborough	Dog
1	2018	Pialba	Dog
2	2018	Tiaro	Dog
1	2018	Urangan	Dog

The above SQL data output is an example of what the de-identified yearly suburb-based animal registration data might look like for consumption while meeting the requirements outlined by the council's open data steering committee.

For dataset 2, the following SQL statement has been produced to develop animal complaint data for release:

```
SELECT COUNT(complaintid) AS number_of_complaints, complainttypedesc, animaldesc,
    complaintsuburbofanimal, EXTRACT(MONTH FROM complaintdate) AS
    complaintmonth,
    EXTRACT(YEAR FROM complaintdate) AS complaintyear
FROM complainttypes INNER JOIN animalcomplaints
    ON complainttypes.complainttype = animalcomplaints.complainttype
    INNER JOIN animals ON animalcomplaints.animalid = animals.animalid
    INNER JOIN breedtypes ON animals.breedid = breedtypes.breedid
    INNER JOIN animaltypes ON breedtypes.animaltype = animaltypes.animaltype
WHERE animalcomplaints.complainttype IN (SELECT complaintType FROM complainttypes
    WHERE severitylevel <> 3)
GROUP BY EXTRACT(YEAR FROM complaintdate), EXTRACT(MONTH FROM complaintdate),
    complaintsuburbofanimal, animaldesc, complaintTypeDesc;
```

NUMBER_OF_COMPLAINTS	COMPLAINTTYPE-DESC	ANIMAL-DESC	COMPLAINTSUB-URBOFANIMAL	COMPLAINT-MONTH	COMPLAINT-YEAR
1	Noise	Dog	Burrum Heads	1	2018
1	Noise	Dog	Burrum Heads	2	2018
1	Roaming	Dog	Tiaro	2	2018
1	Roaming	Dog	Tiaro	4	2018



4.4.10

PostgreSQL
subquery

The above SQL data output is an example of what the de-identified complaint dataset might look like. It meets the requirements outlined by the council's open data steering committee. The scalar function EXTRACT has been used to extract month and year data from the complaint date field.

It should be noted that many built-in functions within different database management systems, including the EXTRACT function, are add-ons, and not ANSI SQL.

As a result of the data outputs produced by the advanced SQL statements, the following data dictionaries have been created to use as guides to develop the output database and form part of the final open dataset documentation.

Table 4.4.1 Data dictionary for Animal Registrations

DATA ITEM	DATA TYPE	VALIDATION RULES	EXAMPLE DATA
registrationyear	INTEGER	Primary Key	2018
animalsuburb	VARCHAR(64)	Primary Key	Nikenbah
animaldesc	VARCHAR(64)	Primary Key	Dog
numberofnanimals	INTEGER	0 < x < 9	3

Table 4.4.2 Data dictionary for Animal Complaints

DATA ITEM	DATA TYPE	VALIDATION RULES	EXAMPLE DATA
complaintyear	INTEGER	Primary Key	2017
Complaintmonth	INTEGER	Primary Key	10
animalsuburb	VARCHAR(64)	Primary Key	Urangan
animaldesc	VARCHAR(64)	Primary Key	Dog
complainttypedesc	VARCHAR(64)	Primary Key	Noise
number_of_complaints	INTEGER	0 < x < 9	7

Serving the FCRC animals registration de-identified open data

The council's open data steering committee has, on advice from its technical staff, decided to separate the output from each open data process to independent databases held externally on its corporate data stores. This decision was made in an effort to add additional protection to the council's internal data stores and not directly connect them to their open data API.

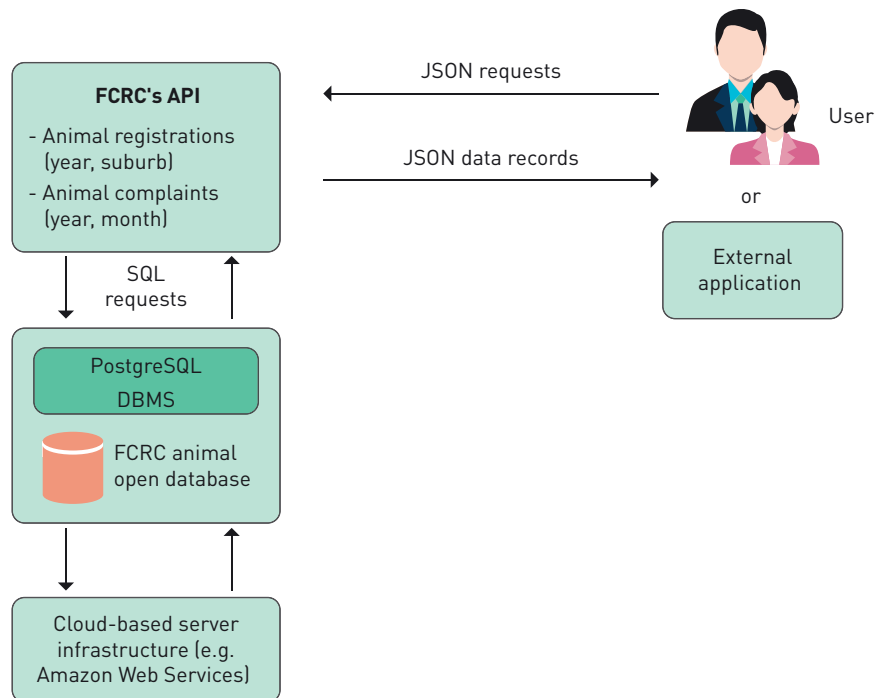


Figure 4.4.10 Illustration of the API application stack and communication processes with external entities

In this section, a simplistic database containing two core tables will be created.

During this process, CREATE, ALTER and DROP statements will be covered. The data output from the previous sections will then be fed into the system using INSERT and UPDATE commands. The DELETE command will also be highlighted during this process. Example SQL SELECT statements used by the API will also be demonstrated to help understand the internal database calls used by the API prior to responding to JSON requests for data.

Data definition elements are used to create, alter or remove structural components of a database. This includes the creation and removal of the database itself, table structures, as well as keys and other constraint settings. While most database environments include non-standard additional DDL features, only core ANSI elements of SQL data definition are explored in this unit.



SQL revisited: CREATE, ALTER and DROP

The CREATE command provides a mechanism to establish databases as well as tables and other objects within a database. To use the CREATE command to create a database, the basic syntax is as follows:

```
CREATE DATABASE fcrcanimalsopen;
```

To use the CREATE command to create a table, the basic syntax is as follows:

```
CREATE TABLE table_name (
  field1 datatype [CONSTRAINTS] [PRIMARY KEY],
  field2 datatype [CONSTRAINTS] [PRIMARY KEY],
  field3 datatype [CONSTRAINTS] [PRIMARY KEY],
  ....
  fieldN datatype [CONSTRAINTS] [PRIMARY KEY],
  FOREIGN KEY (fieldX)
  REFERENCES foreign_table (foreign_key_field)
);
```

The following SQL code examples would be used to create the tables required in the FCRC's open data animals database.

```
CREATE TABLE fcrcanimalsopen.registrationdata (
  registrationyear smallint NOT NULL PRIMARY KEY,
  animalsuburb varchar(64) NOT NULL PRIMARY KEY,
  animaldesc varchar(64) NOT NULL PRIMARY KEY,
  numberofanimals smallint NOT NULL
);
```

This CREATE TABLE statement establishes the core features of the 'registrationdata' table. The three initial fields make up the composite primary key. All fields are required to hold a value in each record. Therefore, all fields have been set to have NOT NULL constraints.

```
CREATE TABLE fcrcanimalsopen.complaintdata (
  complaintyear smallint NOT NULL PRIMARY KEY,
  complaintmonth smallint NOT NULL PRIMARY KEY,
  animalsuburb varchar(64) NOT NULL PRIMARY KEY,
  animaldesc varchar(64) NOT NULL PRIMARY KEY,
  complainttypedesc varchar(64) NOT NULL PRIMARY KEY,
  number_of_complaints smallint NOT NULL
);
```

This CREATE TABLE statement establishes the core features of the 'complaintdata' table. The five initial fields make up the composite primary key. All fields are required to hold a value in each record. Therefore, all fields have been set to have NOT NULL constraints. In both situations, no further constraints or foreign key declarations are made due to the relatively simple requirements of this database. To review further examples of this, refer to section 2.3.

The ALTER command provides a mechanism to edit databases as well as tables and other objects within a database. There are many options available for altering a table. Here are six scenarios.

TO ADD AN EXTRA FIELD (COLUMN)	TO REMOVE A FIELD (COLUMN)
ALTER TABLE table_name ADD field_name datatype;	ALTER TABLE table_name DROP COLUMN field_name;
TO ADD A PRIMARY KEY	TO RENAME A TABLE
ALTER TABLE table_name ADD CONSTRAINT table_primary_key PRIMARY KEY (field1, field2...);	ALTER TABLE name RENAME TO new_name;
TO RENAME A FIELD (COLUMN)	TO SET A FIELD CONSTRAINT AS NOT NULL
ALTER TABLE name RENAME field_name TO new_field_name	ALTER TABLE table_name MODIFY field_name datatype NOT NULL;

The following SQL code snippets are examples for editing the ‘registrationdata’ or ‘complaintsdata’ tables from the ‘frcanimalsopen’ database.

```
ALTER TABLE frcanimalsopen.registrationdata (  
  MODIFY animaldesc varchar(120) NOT NULL;
```

This ALTER TABLE statement modifies the constraints of the ‘animaldesc’ field by increasing the character count. It also ensures the field remains mandatory by declaring its inability to commit NULL values.

```
ALTER TABLE frcanimalsopen.complaintsdata  
  ADD updatedtimestamp datetime;
```

This ALTER TABLE statement adds an additional field to the ‘complaintsdata’ table.
The DROP command provides a mechanism to delete databases as well as tables and other objects from within a database. The syntax for dropping a table or database is shown below.

TO DROP A DATABASE	TO DROP A TABLE
DROP DATABASE database_name;	DROP TABLE table_name;

```
DROP TABLE frcanimalsopen.complaintsdata;
```

This DROP TABLE statement will completely delete the ‘complaintsdata’ table and all data contained within it.

```
DROP DATABASE frcanimalsopen;
```

This DROP DATABASE statement will completely delete the entire Fraser Coast Regional Council animals open database from the database management system.

SQL revisited: insert, update and delete SQL statements

Let’s look at some examples of the INSERT and DELETE statements:

INSERT SQL STATEMENT EXAMPLES	DELETE SQL STATEMENT EXAMPLES
<pre>INSERT INTO frcanimalsopen. registrationdata (registrationyear, animalsuburb, animaldesc,numberofanimals) VALUES (2019,'Pialba','Dog',64)</pre> This example inserts a record of yearly registration summary for dogs in the suburb Pialba into the ‘registrationdata’ table.	<pre>DELETE FROM frcanimalsopen. registrationdata</pre> This example deletes all records from the ‘registrationdata’ table.
<pre>INSERT INTO frcanimalsopen. complaintsdata (complaintyear,complaintmonth, animalsuburb, animaldesc, complainttypedesc, number_of_ complaints) SELECT complaintyear,complaintmonth, animalsuburb, animaldesc, complainttypedesc, number_of_complaints FROM tempImportComplaintTable;</pre> This example inserts new fields from a temporary table that contains new un-entered data.	<pre>DELETE FROM frcanimalsopen. complaintsdata WHERE number_of_complaints > 50</pre> This example deletes records from the ‘complaintsdata’ table where the number of complaints are more than 50.



4.4.12 INSERT statements
4.4.13 DELETE statements
4.4.14 UPDATE statements

Let's look at an example of an UPDATE statement:

```
UPDATE fcranimalsopen.registrationdata
SET animalsuburb = 'Hervey Bay CBD'
WHERE animalsuburb = 'Pialba'
```

In this example, the value of each 'animalsuburb' field in a record has been changed to 'Hervey Bay CBD', but only when the original data was 'Pialba'.

Sample SQL SELECT statements used by an API

As shown in figure 4.4.10, the API has the capability to receive calls from external users. These users can provide optional parameters when required. For the 'registrationdata', users can provide parameter values for the year and suburb. For the 'complaintsdata', users can provide parameter values for the year and month.

These parameter values would be received by the API along with the initial request to provide either registration or complaint data. The the API could then use the parameter values to construct string values of SQL syntax.

The examples of generating SQL within code on the fly demonstrate one known method for completing this task. These methods are potentially prone to security attacks such as SQL injection. Other, more advanced methods, such as the use of SQL stored procedures, can be used in real-life examples. These advanced methods are worth investigating to avoid issues such as SQL injection when developing future digital solutions.

Here is some sample C# code:

```
const string connectionString = "Data Source=(local);Initial
Catalog=fcranimalsopen; UserId = website; Password=DlgItal$S0lut10Ns";

//user-defined function that is called and passes two values - Month
and Suburb. These values have previously been provided from the user. It
returns a structure called a sqldatareader that returns a recordset of
values.

SqlDataReader GetRegistrationData(string MonthFromUser, string
SuburbFromUser)
{
    //develops a simple query of fields from complaintdata table where month
    and suburb matches ones selected by user

    string queryString = "SELECT complaintyear, complaintmonth, animalsuburb,
    animaldesc, complainttypedesc, number_of_complaints FROM
    dbo.complaintsdata WHERE animalsuburb ='" + userSelectedsuburb + "' AND
    complaintmonth ='" + userSelectedmonth + "';";

    using (SqlConnection GEConnection = new SqlConnection(connectionString))
    {
        // Set a new data (SQL) command for the database
        SqlCommand command = new SqlCommand(queryString, GEConnection);

        // Open the Database Connection
        GEConnection.Open();
```

```

        // returns an Executed SQL retrieval of data records from database
        table
        Return command.ExecuteReader();
    }
}

```

In this example, a string variable called 'querystring' is used to construct a full statement that can be triggered to extract filtered data from the table.

To further review SQL before the external examination, it is recommended students review the SQL concepts in section 2.3.



TEST YOUR KNOWLEDGE

- 1 What is the purpose of the Australian Privacy Principles?
- 2 Which Australian Privacy Principle describes how information can be given to others? What are the requirements?
- 3 When creating an open data source from an organisation's data, which Australian Privacy Principles provide clear advice about how to deal with sensitive personal information in this scenario?
- 4 What is the difference between closed, shared and open data?
- 5 Describe some of the methods used to protect stored and transmitted sensitive data.
- 6 What are the potential economic benefits of open data?
- 7 What are the potential social benefits of open data?
- 8 What is the purpose of SQL data definition commands?

Use the sample internal FCRC animal registration and complaints database to generate SQL statements to solve the following tasks:

- 9 Find the animal name, breed description and latest registration year of each animal in the system that is active.
- 10 Find the number of animals registered to each owner. Include the owner names in the results.
- 11 Find the animal with the most number of complaints against it.
- 12 Determine if any animals that are active have not been re-registered in 2018.
- 13 Provide a count of the range of different breeds for each animal type.
- 14 Find the owner who owns the greatest number of cats in the council area.
- 15 Find which suburb has the least number of dogs associated with it.





APPLY YOUR KNOWLEDGE

- 1 FCRC's open data steering committee has been queried about the correctness of the open dataset created from its animal registration and complaint data. The specific enquiry relates to the number of active breeds within the council area.
- 2 FCRC's tourist information centre provides advice to travellers on a daily basis at a number of its manned sites. During enquiries, staff and volunteers collect personal information as shown below:

Your task is to test the quality of the data in the external open data source against the data contained within the internal data source. Generate two SQL statements (one within each database) that will confirm alignment of active breeds within the council area.

NAME	ADDRESS	LOCATION	POSTCODE	STATE	COUNTRY	ENQUIRY TYPE	INFORMATION GIVEN

The open data steering committee has asked for advice on how this information could be adjusted to become an open data source. Provide advice on potential adjustments that meet the council's open data processes and develop one or more SQL statements to potentially generate a sample open data set for testing.

- 3 Review the case study relating to London transportation.

Evaluate the effectiveness of this open data-driven initiative against economic, social and personal perspectives.



Weblink

GLOSSARY

Algorithm A step-by-step procedure required to solve a problem. For example, to find the largest number in a list of positive numbers: • note the first number as the largest; • look through the remaining numbers in turn, and if a number is larger than the number found in the first step, note it as the largest; • repeat this process until complete – the last noted number is the largest in the list. An algorithm may be described in many ways. Pseudocode is often useful in visualising an algorithm.

Analyse Dissect to ascertain and examine constituent parts and/or their relationships; break down or examine in order to identify the essential elements, features, components or structure; determine the logic and reasonableness of information; examine or consider something in order to explain and interpret it for the purpose of finding meaning or relationships and identifying patterns, similarities and differences.

Application programming interface (API) A code library with defined functions, inputs and outputs that allows one programmed component to interact with a component written by another developer.

Asynchronous An exchange that does not need to wait for a reply before moving to another activity or process, e.g. email messages.

Cardinality Grouping rules in a relational database structure that define the number of relationships permitted between related records.

Cartesian join An undesirable join result that maps every row in one table to every row in a second table, ignoring the relationships between records in each table.

Compiling A translation process to convert a software program from human instructions written in programming syntax to machine-readable instructions in binary format. The compiled program is stored as an executable file.

Composite key A primary key that requires two or more fields to uniquely identify a record in the table.

Computational thinking A problem-solving method that involves various techniques and strategies that can be implemented by digital systems. Techniques and strategies may include organising data logically, breaking down problems into parts, defining abstract concepts, and designing and using algorithms, patterns and models.

Constraint A technical, legal, economic or environmental restriction that applies to a digital project.

Constructor A function that defines all the data properties of a class.

Convergent thinking Organising and refining ideas or information to identify common themes, features or issues.

Crowdsourcing Collecting data from a large number of volunteers using the internet or social media.

Data anomalies Problems that occur when inserting, updating or deleting data in a relational database that does not have an optimal normalised structure.

Data integrity The requirement to maintain the quality and consistency of data stored collectively so that it continues to have meaning and value.

Data reliability The ability to trust that a data source accurately represents the facts, descriptions or actions it purports to represent.

Data redundancy The unnecessary duplication of data elements within a data store, which may lead to problems with data validity and integrity.

Data validation Checking that input data is in the expected format to reduce or prevent user errors.

Data validity When data (or information) meets defined formats, rules and criteria in order to maintain integrity and reliability.

Data verification Checking that input data matches related data in an external data source to appraise whether the data is accurate.

Denormalisation A strategy used to combine previously separated datasets into a single table structure by adding related data and possibly creating redundancy. The process is most commonly used to simplify the structure of data prior to exchange, or to optimise frequently accessed read-only datasets.

Design thinking The use of strategies for understanding design problems, visualising and generating creative ideas, and evaluating those ideas that best meet the criteria for success. Students use both convergent and divergent thinking skills: divergent thinking supports creativity and the generation of a range of ideas, and convergent thinking supports the selection and generation of a preferred solution.

Digital disruption The transformation that occurs when innovative digital solutions or emerging digital technology significantly impacts a process, product or activity.

Divergent thinking Brainstorming that considers as many different ideas as possible.

Domain integrity The need for all values stored in a field or column of a database to be constrained to a specific format such as a data type, length or naming convention.

Encapsulation The process of creating a class of properties and methods.

Federating data The practice of combining related data records from different databases into a single record.

Foreign key A field that is a primary key in a different table and can be used to link additional data from a corresponding record in another table.

Function A program module that can accept input and return output. Input, if required, is passed to the function using parameters, and output can be returned, if required, with the return keyword.

Functional requirement One of the specific actions and activities that a digital solution needs to be able to do in order to fulfil its primary purpose.

Global variable A variable accessible throughout an entire software program.

Heterogenous Components in a digital solution that use different operating systems, data formats and/or network protocols.

Idempotent The mathematical capability of applying repeated operations without changing the original value. When applied to RESTful services, it means that the operation is safe, however many times it is executed, because it will not change the value of the underlying data.

Innovation The art of finding a new and better solution to an existing problem.

Intelligent system One that includes hardware components with sensing and actuating technology and solutions that focus on automated systems designed for a user experience. These technologies allow students to explore how computational

processes respond to given inputs (e.g. environmental information such as light or sound) to produce specified outputs. Feedback occurs through tight loops, allowing for trial-and-error investigations.

Instantiate Use the class blueprint to create a copy of an object.

Interpreted code Programming syntax that is translated line by line into machine-readable instructions at run time.

JSON JavaScript Object Notation. A common data exchange format used to structure data in key-value pairs. Although the format is based on JavaScript, it is supported by a range of programming languages.

Local variable A variable with a limited scope, such as one within a function.

Metadata Data about data.

Mobile application The type of software application that runs on a mobile device, such as a smartphone or tablet computer, and often provides users with similar services to those accessed on desktop computers.

Non-functional requirement One of the features of a solution that specify how it should work, rather than what it does. This can include aesthetics, accessibility and efficiency targets.

Normalisation The process of restructuring tables in a relational database to obtain an optimal structure known as optimal normal form (ONF).

Objects A way of organising data (properties) and functions (methods) into an advanced data structure.

Parameter A data value that is passed to a function as input.

Persona A profile of a typical target user.

Prescribed criteria The desired qualities or features of a digital solution, as defined by an external party such as a teacher or client.

Primary key A field or set of fields in a table that uniquely identifies a single record in the table.

Proto-persona A profile of an imagined user. Proto-personas are less accurate than personas but easier to create.

Protocol One of a set of generally accepted standards or 'rules' that govern relationships and interactions between, and within, information systems.

Pseudocode A method of writing algorithms as short sentence fragments. Pseudocode describes the logic of a computer task using code-like keywords common to a range of programming languages.

Public key encryption A method of encryption that uses a public key to encrypt a secret message and a private key to decrypt it.

Representational State Transfer (REST) A framework used to exchange data using an Internet-accessible data resource instead of an advertised data service. The main characteristics of a RESTful exchange are that it relies on standard HTTP requests using embedded SSL internet security protocols, and that each request is processed independently (i.e. it is stateless).

Self-determined criteria The desired qualities or features of a digital solution defined by a member of the project team such as a student.

Semantic tags HTML5 tags, such as <header>, <nav> and <aside>, that are used to add meaning to content.

Serialisation The process of reorganising relational data into a linear structure that can be stored as a series of bits.

Structured data Data organised into fields and records, such as columns and rows in a database table or forms and tables on a web page.

Symmetric encryption A method of encryption requiring the sender and receiver to use the same secret key.

Synchronous A timed exchange in which each side takes turns and waits for a response before moving to the next activity or process. For example, most online payment systems wait for acknowledgement that a credit card transaction is successful before processing an online order.

Syntax A set of rules that defines the structure of statements in a computer language.

Systems thinking A holistic approach to the identification and solving of problems, where parts and components of a system, their interactions and interrelationships are analysed individually to see how they influence the functioning of the whole system. This approach enables students to understand systems and work with complexity, uncertainty and risk. Systems thinking also involves understanding the interdependence between systems and how a change or output from one system can affect another, and how this affects larger systems such as the economy and society.

Uniqueness constraint A rule applied to a field in a relational table that prevents duplicate field values from being stored.

Useability How easy a product is to use.

Web application Client-server software applications in which the client-user interface runs in a web browser, e.g. webmail, online retail sales.

INDEX

- abstraction 4, 10, 42
- accessibility (interfaces) 26–8, 231
- accessibility guidelines 279–80
- accessible images 27–8
- accessible multimedia 28
- accordions 317
- acquisition of data 112–13
- action priority approach 282
- actuators 207
- AES (Advanced Encryption Standard) 292
- aesthetics of a digital solution 234–5
- aggregate functions (SQL) 166–7
- agile project management 251–2
 - principles 251
 - user stories 252
- AJAX (Asynchronous JavaScript and XML) 299, 304–5
 - use with jQuery 322
 - use with XMLHttpRequest 322
- algorithm control structures 37
- algorithm walk-throughs 45
 - Euclid's tiles 48–52
 - Fibonacci bubbles 45–8
- algorithmic thinking 10
- algorithms 2, 38–9, 213
 - for data-driven solutions 130–1
 - developing 39–44
 - for objects and classes 58–9
 - for one-time pad cipher 336–8
 - for polyalphabetic ciphers 330–3
 - for programmed components 244–5
 - good and bad examples 256
 - removing ambiguity 310, 320, 321
- alignment (CARP) 24
- alignment (visual communication) 99–100, 235
- ALTER command (SQL) 154–5, 371–2
- analysing a data exchange problem 310–13
- annotated site maps 228
- APIs (application programming interfaces) 204, 370
 - using SQL SELECT statements 373–4
- application systems 123
- application tokens 294
- arithmetic (operation) 41
- array of objects example 57–8
- arrays 54, 306
- ASCII Aliens (interactive media case study)
 - exploring the problem 14–16
 - mind map 15
 - programmed solutions 75–85
 - class diagram 76
 - hierarchy chart 78
 - HTML 84
 - JavaScript 84
 - pseudocode 79–84
 - selecting an IDE 16
 - user interfaces 31–2
- ASCII tables, use in making ciphers 327, 328
- assignment (operation) 41
- associative arrays 306
- asymmetric encryption 339, 346–50
 - HTTPS 347, 359
 - RSA 347–50
 - use in public and private keys 346–7
- asynchronous exchange 286
- attributes (ER diagrams) 133
- audio files, accessibility 28
- Australia Card, 1980s 184–5, 186
- Australian Privacy Principles 182, 185, 186, 281, 355
 - Principle 1: open and transparent management of personal information 356
 - Principle 6: use or disclosure of personal information 356
 - Principle 11: security of personal information 356
- Australian Red Cross blood service data breach case study 289–90
- authentication 290, 293–4
 - application tokens 294
 - digital signatures 293–4
- automation technologies 7
- availability (data exchange) 290
- backlog 253
- backup 117
- balance (visual communication) 102, 235
- BBC Microbit 8
- behaviour-over-time graphs 210
- best-effort service QoS model 299, 300
- binary files 268
 - reading and writing 269–70
- block ciphers 338, 339
- Blowfish 292
- Boolean data type 37, 110, 111, 112
- box office rewards problem 214
- brainstorming 218, 219, 224, 227, 228, 286, 311
- breakpoints 23
- bubble sort 54–5
- burn down 253
- burn up 253
- Caesar cipher
 - applicable to computers 328
 - breaking the 328, 338
 - history 326
 - practical example 327
 - use with polyalphabetic ciphers 329
- calculations and functions (SQL) 164–7
- cardinality 109, 133–4, 149
 - Garden Express case study 136–7
- cardinality lines 133, 134
- CARP design principles 24
- casual loop diagrams 210
- categorised data 271
- 'char' data type 110
- characters and strings (data type) 110, 111
- check digit 262
- checksums 290, 294–5
- chilli game 38
- CIA triad 290, 291
- ciphers
 - block vs stream 338–9
 - Caesar 326–8
 - Feistel approach 339
 - one-time pad 336–8
 - polyalphabetic 328–36
 - quality of 338–46
 - simplified DES (S-DES) cipher 339–46
 - symmetric vs asymmetric 339, 346–50
 - Triple DES 262, 350–1
- class 55
- class constructor 55
- class diagrams 76
- cleaning and merging data 270–2
- cleaning data 240, 271
- cleaning strategy, developing 272–5
- Clever web application, case study 360–2
- closed data 353
- closed systems 210
- closing the database connection 266
- code editors 44
- code indentation 40, 46
- code libraries, encryption via 350–1
- coded components, defining 228–30
- coding for database interactions 263–6
 - close the connection 266
 - establishing a connection to the database 263–4
 - processing SQL results 265–6
 - sending an SQL request 265
- coding for file input/output 266–70
 - file access 267–8
 - file structure 268
 - reading a text file 268–9
 - reading and writing binary files 269–70
- coding for user interactions 259–62
 - communicating effectively with the user 259–60
 - validating user data 261–2
- colour (visual communication) 97–8, 235
- combining data sources 239–40
- commodity, information as a 182–3
- communicating effectively with the user 259–60
 - presenting information to the user 260
 - processing user input 260
 - providing feedback to the user 260
 - waiting for user input 260
- comparison operators 160
- compiling (instructions) 36
- computational thinking 9–10, 11, 250, 255
- computer operations 41
- computer systems 123–4
- computer variables 37, 39, 54
- concept maps 210–11, 213, 214, 257, 312
 - tools suitable for creating 212
- condition (operation) 41
- confidentiality (data exchange) 290

- confirmation codes 262
- connected circle 210
- connecting lines (relational schema) 149
- consent prompts 28
- constraints 223
- consumer loyalty cards 182–3
- context data flow diagrams 242
- context diagrams 125
- contrast (CARP) 24
- contrast (visual communication) 100, 235
- control mechanisms
 - intelligent systems 209
 - mobile applications and interactive media 205
 - web applications 198–200
- control structures 41, 53
- controlling digital devices 35–6
- convergent thinking 10–11, 218
- copyright considerations
 - and intellectual property 281
 - for interactive media 205–6
- counted loop 53
- CREATE command (SQL) 153–4, 370–1
- Creative Commons 281
- cross-platform origin errors 323
- CSIRO Syropods 206
- CSS (cascading style sheets) 68
 - grids 317
 - !important tag 199
 - Koala Tracker case study 200
- CSV files 268, 272
- cybersecurity 284
 - Australian Red Cross blood service
 - data breach 289–90
 - encryption and decryption 291–2, 327–38
 - internet transmission principles
 - 296–308
 - processes for secure data storage and exchange 290–6
 - see also* data security
- data 88–90
 - de-identification of organisational data
 - 187–8
 - explosion in 181
 - published with degrees of openness
 - 353–4
 - security impacts 182
 - sources and uses of 91
 - three-layer approach to 94–6
 - visual communication of 97–103, 235–6
 - working with 94–6
- data acquisition 112
 - ownership of data 113
 - time factors 112–13
- data anomalies 114–15
- data collection and privacy 183–6, 223
- data components
 - data exchange solutions 313–15
 - digital solutions 236–40
 - web application (tier 3) 196
 - data compression 298
 - data definition elements 370
 - data dictionaries 149, 150, 198, 200
 - data disposal 186–7
 - data-driven solutions
 - application, presentation and interactive components 105
 - components across systems 104
 - data flows 124–30
 - data management and storage 105
 - developing algorithms for 130–1
 - evaluating 188–91
 - flat files as databases 105, 106, 237–8
 - relational databases 105, 106–12, 131–45, 237
 - systems thinking for 122–4
 - working with data: SQL 150–8
 - working with relational schemas 148–50
 - data-driven systems thinking 122–4
 - data elements 89
 - data encryption *see* encryption
 - data errors, common 271–2
 - data exchange 123, 284
 - evaluating impacts of 353–74
 - OpenActive case study 285–8, 310–21
 - and secure data storage, processes 290–6
 - security issues 359
 - selecting an appropriate approach 322–5
 - data exchange algorithm 320–1
 - data exchange methods (over the Internet) 303–8
 - AJAX 304–5
 - JSON 305
 - RESTful services 304
 - XML 306–8
 - data exchange problem, analysing 310–13
 - data exchange solutions, developing 310–25
 - data exchange system, elements 285–6
 - data facts 89
 - data flow diagrams (DFD) 124, 198, 199, 227, 241
 - characteristics 124–5
 - context 242
 - decomposing different functions of the system 242–3
 - examples 125–6
 - Garden Express case study 127–30
 - IPO charts used in conjunction with 130–1
 - numbering system for processes in 241
 - processes that could be documented 125–6
 - rules for creating 125, 243
 - top-level context for OpenActive 313
 - data formats 105–6, 236–7
 - selecting 237–9
 - data hashing 116–17, 290, 295–6
 - data–information–knowledge–wisdom model 90
 - data integrity 106, 113–14
 - data interface, developing 315–17
 - data loss 28
 - data management and storage components 104, 105
 - data management issues 112
 - acquisition 112–13
 - data anomalies 114–15
 - data integrity 113–14
 - security and protection 115
 - data management security solutions
 - backup and disaster recovery processes 117
 - encryption 116–17
 - user-level access 116
 - data mining 182
 - data needs and requirements, exploring 132–3
 - data patterns 262
 - data privacy 183–7, 223, 281
 - data processes and system interactions
 - mobile applications and interactive media 205
 - web applications 198
 - data relationships, identifying 109, 134, 137, 274–5
 - data requirements, defining 226–7
 - data retention 186–7
 - data security 182
 - of shared data exchange 359
 - see also* cybersecurity
 - data security methods 326
 - Caesar cipher 326–8
 - one-time pad cipher 336–8
 - polyalphabetic ciphers 328–36
 - data sources
 - combining 239–40
 - merging 240
 - data stores 90, 96
 - data structures 37, 54–6, 306
 - data systems 124
 - data theft 115
 - data transfer, Big-Endian and Little-Endian 296
 - data transmission 297–301
 - data types 37
 - in PostgreSQL database management system 111–12
 - problems of selecting incorrect 112
 - used in databases 110–11
 - data validation 261–2
 - data verification 261
 - database interactions, coding for 263–6
 - database management systems 105
 - SQL variants in 118, 152
 - see also* relational database management systems

- Database .NET (visual editor) 152
 - table creation 150
- database(s) 95, 96, 105
 - closing the connection to 266
 - data types 110–11
 - establishing a connection to 263–4
 - flat files as 105, 106, 237–8
 - processing SQL results 265–6
 - relational 95, 105, 106–12
 - sending an SQL request to 265
- datasets 91
 - exploring 272
 - identifying data relationships in 274–5
 - merging 272, 273, 275
- date and time (data types) 110, 111, 112
- DDL (Data Definition Language) SQL 153–5, 272, 370–2
- de-identification of organisational data 187–8, 354
 - advanced SQL techniques 365–8
 - Fraser Coast Regional Council animal registry and complaints data 362–74
 - generating de-identified data 368–9
 - Heritage Health Prize case study 357
- debugging tools 57
- ‘decimal’ (data type) 110, 111
- decomposition
 - definition 10
 - of a digital problem 3–4, 241–3
 - of a digital system 7–8
- defining the problem (digital solutions) 223
 - defining the coded components 228–30
 - defining the data requirements 226–7
 - defining the user interface 224–6
 - evaluating the requirements 230
 - mind maps for brainstorming 224
- DEFLATE 298
- DELETE SQL statements 170, 372
- denial-of-service attack 284
- DES cipher 339
- design principles 99–103, 235
- design thinking 10–11, 218
 - group activities 219–22
 - guidelines 218
 - use in a digital technologies context 218–22
- desk checking an algorithm 43–4, 46, 332–3
- developing (digital problem-solving process) 4
- developing algorithms 39
 - desk checking the algorithm 43–4
 - developing an algorithm idea 40–3
 - exploring and unpacking the problem 39–40
- developing cleaning strategies 272–3
 - documenting the table schemas 273–4
 - evaluating the success or a merge or join 275
 - identifying the data relationships 274–5
 - merging datasets 273
- developing data components (data exchange solution) 313–14
- developing data components (digital solutions) 236
 - combining data sources 239–40
 - common data formats 236–7
 - selecting an appropriate data format 237–9
- developing data exchange solutions 310
 - analysing a data exchange problem 310–13
 - developing the data components 313–15
 - developing the data interface 315–17
 - developing programmed components 317–21
 - selecting an appropriate data exchange approach 322–5
- developing data interfaces (data exchange) 315–16
 - managing presentation of dynamic data components 316–17
- developing innovative solutions 217
 - defining the problem 223–30
 - developing the data components 236–40
 - developing programmed components 241–5
 - developing the user interface 230–6
 - refining the solution 245–6
 - using design thinking 218–22
- developing programmed components (data exchange) 317–19
 - developing algorithms 320–1
 - investigating programming environments 319–20
- developing programmed components (digital solutions) 241
 - decomposing the problem 241–3
 - developing algorithms for programmed components 244–5
- developing user interfaces (digital solutions) 230–1
 - elements and principles of visual communication 235–6
 - low-fidelity prototypes 233–4
 - responsive interfaces 233
 - useability 231–3
- development environment, setting up 67
- differentiated service (QoS model) 299
- Digest Access Authentication 293
- digital disruptions 217
- digital environment, establishing 254–5
- digital innovation 217
 - using design thinking in 218–22
- digital problem-solving process 3–5
 - developing 4
 - evaluating 5, 188
 - exploring 3–4, 5–20
 - generating 4–5
- digital problems, what are they? 2
- digital prototype creation 250
 - establishing the digital environment 254–5
 - generating an interactive data-driven digital prototype 255–75
 - project management approaches 250–4
- digital signatures 293–4
- digital solutions, evaluating 277–82
- Digital TV Antennas app, case study 358–9
- disaster recovery processes 117
- disposal of data 186–7
- divergent thinking 10, 218
- DML (Data Manipulation Language) SQL 155–71, 372–4
- domain integrity 240
- DROP command (SQL) 155, 372
- duplicate data 271
- ECMA International 66
- ECMAScript 53–4
- economic benefits of open data 354, 358
- EDGE (problem-solving process) 3–5, 45
- effectiveness (interfaces) 24–5, 232
- elements of intelligent systems 207
 - control mechanisms 209
 - data processes and system interactions 208–9
 - hexapod robot 207
 - robot interactions 207–8
 - terrain types used with the robot 208
- elements of mobile applications and interactive media 203–4
 - copyright considerations for interactive media 205–6
 - data processes, system interactions and control mechanisms 205
 - user interactions 204–5
- elements of visual communication 97–9, 235
- elements of a web application 195
 - control mechanisms 198–200
 - data processes and system interactions 198
 - three-tier architecture 195–6
 - user interactions 196–7
- encapsulation 55
- encryption 116, 290
 - how hashed password authentication works 116–17
 - shared data exchange 359
- encryption algorithms, properties 339
- encryption and decryption 291–2, 326
 - one-time pad cipher 336–8
 - polyalphabetic ciphers 328–36
 - quality of ciphers 338–46
 - symmetric vs asymmetric 339, 346–51
 - using the Caesar cipher 327–8
 - via code libraries 350–1
- Endian protocol 296

- Enigma Machine 326, 329
- entities (ER diagrams) 133
- entity-relationship (ER) diagrams 95, 132, 133–4
 - developing 135
 - Garden Express case study 135–8
 - OpenActive data model 287
- Euclid's tiles (algorithm walk-through) 48–52
 - algorithm 50
 - output and code 51–2
- European Computer Manufacturers Association (ECMA) 52–3
- evaluating (digital problem-solving process) 5, 188
- evaluating data-driven solutions 188
 - during the investigation of a problem 188–9
 - evaluating solutions 190–1
 - while developing solutions 189
 - while generating solutions 190
- evaluating digital solutions 277–82
 - data privacy 281
 - establishing evaluation criteria 277
 - evaluating impacts 279–81
 - justifying and prioritising recommendations 282
 - testing the solution 277–9
 - using self-determined criteria 281–2
- evaluating impacts 279
 - accessibility guidelines 279–80
 - data privacy 281
 - intellectual property and digital copyright 281
- evaluating using self-determined criteria 281–2
- evaluation 10
- evaluation criteria, establishing 277
- event listeners 56
- exploring (digital problem-solving process) 3–4
 - case studies 11–20
 - combining design, computational and systems thinking 11
 - computational thinking 9–10, 11
 - design thinking 10–11
 - key steps in exploring the problem 5–6
 - systems thinking 6–8, 11
 - using mind maps 5–6
- exploring datasets 272
- external layer view of data 96
- facial recognition 11
- Feistel cipher approach 339
- Fibonacci bubbles (algorithm walk-through) 45–8
 - adding gravity 64
 - algorithms for objects and classes 58–60
 - analysing the human need or want 45
- array of objects 57–8
 - create an event listener 59
 - decompose the problem into smaller parts 45
 - define the data structures used 58
 - desk check 46
 - develop the algorithm idea 46
 - develop ideas for the user interface 46
 - evaluate the solution 60
 - generate the code 46–8, 59–60, 61
 - identify requirements 45
 - making the bubbles bounce 62–4
 - p5.js code so far 63–4
 - pseudocode so far 62
 - recognise constraints 46
 - recommending improvements 61–2
 - restate the problem succinctly 45
 - show and move each bubble object in the array 59
- fields (relational database) 106
- file access 267–8
- file input/output, coding for 266–70
- file structure 268
- first normal form (1NF) 139, 140
- Fitts's law 99
- flat files as databases 105, 106, 237–8
- flexible vs. non-flexible algorithms 41
- float (data type) 110, 111
- flowchart symbols 40, 211
- flowcharts 40, 42, 46, 213
 - post-tested and pre-tested 42
- FOR loop 53
- foreign keys 108, 109, 150
- form (visual communication) 98, 235
- form-processing technologies 113
- forms (for data entry) 91, 112–13
- Fraser Coast Regional Council animal registry and complaints data 362–3
 - advanced SQL techniques to de-identify and create open datasets 365–8
 - Council open data process 363
 - generating de-identified data 368–9
 - mind map exploring the open data problem 363
 - scenario data 363–4
 - serving the FCRC animals registration de-identified open data 370–4
 - what must be de-identified 364
- Fruit Ninja mobile game case study 201–2
 - copyright considerations 205–6
 - elements 203–6
- FTP (File Transfer Protocol) 302–3
- full stack development 67
- functional requirements 223
 - testing they have been met 277–8
- Garden Express (web-based context case study) 92
 - background 92
 - cardinality 136–7
- checking compliance with first, second and third forms 143–5
- constraints 92, 93
- customer billing address update form 174–9
- data flow diagrams 127–30
- data needs and requirements 135
- defining the problem 92–3
- developing relational database 135–9
- displaying a data-driven product list 172–4
- entity-relationship diagrams 135–8
- functional requirements 92–3
- non-functional requirements 93
- problem mind map 93
- SQL SELECT statements 159–63
- generating (digital problem-solving process) 4–5
- generating an interactive data-driven digital prototype 255–6
 - cleaning and merging data 270–2
 - coding for database interactions 263–6
 - coding for file input/output 266–70
 - coding for user interactions 259–62
 - developing a cleaning strategy 272–5
 - generating readable modular programs 256–8
- generating de-identified data 369–70
- geo-targeting 6
- geospatial data 272
- global variables 50, 58
- glossary 376–7
- golden ratio 98–9
- grids 317
- grouping (GROUP BY clause) 167–8, 365–7
 - with HAVING clause 168, 367
- guaranteed service (QoS model) 299
- hardware 2
- hardware components, of a digital solution 7
- hardware systems 123
- hardware view of a programmable board 8
- harmony (visual communication) 103, 235
- hashed password authentication 116–17
- hashing 116–17, 290, 295–6
- HAVING clause 168, 367
- Heritage Health Prize, de-identification of data, case study 357
- heterogeneous collection 298
- hexapod intelligent systems 206–10
- hiding data 28
- hierarchical information systems 105
- hierarchy (visual communication) 100–1, 235
- high-level languages 36–7, 52
- history of programming languages 52–3
- HTML
 - basic understanding of 44
 - body section 69
 - Fibonacci bubbles program 47
 - head section 68

- HTTP (Hypertext Transfer Protocol) 301
 - methods 301–2
 - protocol standard 293
- HTTPS 292, 302, 359
 - asymmetric encryption 347
- human language to machine language 35–6
- human systems 123
- icons 28
- identification card 184–5
- identifying patterns 10
- identity theft 115
- IF statement 53
- IF–THEN–ELSE statements 37, 41
- The Imitation Game* 329
- impacts of storing and exchanging data 353–4
 - de-identification of organisational data 354, 357
 - generating de-identified and aggregated data 362–74
 - impacts of open data 357–9
 - privacy in Australia 355–6
 - security of shared data exchange 359–62
- important tag 199
- information 88, 90
 - as a commodity 182–3
- information integrity 93
- information reliability 94
- information social evolution 183–6
- information systems
 - components 105
 - different forms 105
- information validity 94
- innovation
 - definition 217
 - digital 217
 - using design thinking in a digital technologies context 218–22
- innovative digital solutions 193
 - developing 217–47
 - exploring innovation 194–215
- INSERT SQL statements 170, 372
- instantiated (object) 55
- integer/whole number (data type) 110, 111
- integrated development environments (IDE) 16, 45
 - for generating digital solutions 255
- integrity (data exchange) 290
- integrity (databases) 106, 113–14
- intellectual property and digital copyright 281
- intelligent system context, data-driven
 - system components 104, 105
- intelligent systems
 - coded components 229
 - elements of 207–9
 - Syropods (robots) 206, 207, 209
- intelligent systems multiplayer game case study *see* Microbomb intelligent systems
- interaction sequence diagrams 229
- interactive media case study *see* ASCII Aliens
- interactive media solution
 - data format 239
 - see also* mobile application and interactive media
- interfaces *see* user interfaces
- internal layer view of data 94
- International Organization for Standardization (ISO) 66, 296
- Internet Engineering Task Force (IETF) 297, 298
- Internet protocols 301–3
- Internet transmission principles 296–7
 - Internet protocols 301–3
 - methods for exchanging data 303–8
 - standards bodies 296–7
 - transmitting data 297–301
- interpreting (instructions) 36
- interviews 225–6
- invasion of privacy 115
- IPO (input, process and output) chart 2, 8, 9, 39, 328
 - ambiguous 258
 - for case studies 13
 - in conjunction with data flow diagrams 130–1
 - encryption example 330–1
 - Euclid's tiles 48
 - Fibonacci bubbles 45
 - OpenActive data exchange 320
- iteration 37, 41, 53
- JavaScript 37, 47, 52, 53
- jitter 299
- jQuery 319, 322
- JSON (JavaScript Object Notation) 305, 313–14, 319, 370
- justifying and prioritising recommendations 282
- key fields 240
- key value pairs 306
- keys (ciphers) 326
- keys (relational databases) 107–8
- knowledge 90
- Koala Tracker web application case study 194–5
 - control mechanisms 198–200
 - data dictionary 198, 200
 - data flow diagram 198, 199
 - error feedback 200–1
 - user interface 196–7
- latency 298–9
- learnability (interfaces) 24–5, 232–3
- LIKE clause 160–1
- line (visual communication) 98, 235
- local variables 50, 58
- logic components (web application tier 2) 196
- logic errors 55
- logic operators 53
- logic thinking 10
- logical layer view of data 94–5
- logical operators 159–60
- lookups 262
- looping 41
- lorem ipsum text 246
- low-fidelity prototypes 233–4, 246
- loyalty cards 182–3
- MAC (message authentication code) 294
- malware 115
- many-to-many relationship 109, 134, 137, 274–5
- many-to-one relationship 274, 275
- mathematical operations (SQL) 164–7
- MD5 hash algorithm 116–17, 295
- meaningful names 39–40
- media query 71–2
- merging data/datasets 271, 272, 273, 275
- metadata 186, 187
- Microbomb intelligent systems (case study)
 - exploring the problem 16–20
 - mind map 19
 - user interfaces 32–3
- Microsoft Access 118, 152
- Microsoft SQL Server 117, 152
 - data types 110
- mind maps 5–6, 188–9, 224
 - for case studies 13, 15, 19, 93, 286, 310–11, 363
- mismatched data representations 271
- mobile application and interactive media
 - case study *see* Fruit Ninja
 - coded components 230
 - elements 203–6
- mobile applications
 - data format 238–9
 - involving structured data 189–90
- mobile context, data-driven system
 - components 105
- mobile device data sources 182
- modern ciphers 338–46
- modular pseudocode 43
- modularisation 10, 41–2
- Moore's Law 338
- MoSCoW method 230, 252, 313
- Mozilla Foundation 297
- MyGov website, 2010s 185
- MySQL Server 117, 152
- National Broadband Network (NBN) 300
- navigation bars 69–72

- nested control structure 320, 321
- Nested IF statements 53
- net neutrality 300
- network information systems 105
- network latency 298–9
- network systems 123–4
- Nielsen's useability heuristics 279
- Node.js 196
- non-functional requirements 223
- non-modular pseudocode 43
- normalisation process 139–42
 - checking, Garden Express case study 143–5
 - first normal form (1NF) compliance 139, 140
 - second normal form (2NF) compliance 139, 141
 - third normal form (3NF) compliance 139, 141–2
- NULL values 149, 150, 158
- numbered concept maps 213
- OAuth (Open Authentication) 294
- object properties 56
- object-oriented information systems 105
- object-oriented programming languages 230
- objects 55–6
- OCR (optical character recognition) 113
- one-time pad cipher 336–8
 - algorithm 336–7
 - encryption and decryption 337–8
- one-to-many communication 17
- one-to-many relationship 109, 134, 274, 275
- one-to-one relationship 109, 134, 274
- online surveys 224–5
- 'op' notation 149
- open data 91, 353, 354
 - economic benefits 354, 358
 - Fraser Coast Regional Council animal registry and complaints data 362–74
 - impacts of 357–9
 - preparing 354
 - social benefits 354
- open systems 210
- Open Systems Interconnection (PSI) 296
- model 297
- OpenActive data exchange case study 285, 325
 - brainstorming mind map 311
 - concept map 312
 - constraints and requirements of the solution 286
 - data components development 313–15
 - data exchange algorithms 320–1
 - data interface 315–16
 - data model 287–8
 - elements 285–6
 - MoSCoW table 313
 - programmed components development 317–19
 - top-level context DFD 312–13
- OpenID 294
- openness of data 353–4
- OpenPGP 292
- operating systems 123
- Oracle 118, 152
- ORDER BY clause 169
- organisational data, de-identification of 187–8, 354
- organisational goals 281
- ownership of data 113
- pair programming 258
- paper walk-throughs 233–4
- parameters 58
- partial matches (searches) 160–1
- passwords 116
 - authentication 116–17
- personal goals 281
- personal information, privacy issues 355
- PHP (server-side scripting language) 196
- pitching an idea 246
- polyalphabetic ciphers 328–36
 - algorithm desk checks 332–3
 - algorithms for 330–3
 - code example 333–6
 - cracking the cipher 330
 - encryption and decryption 331–2
 - generation 329
- portability (of interfaces) 23
- post-tested iteration in a flowchart 42
- post-tested loop 53
- PostgreSQL 118, 152, 166
 - data types 111–12
 - and Database .NET 152
- pre-tested iteration in a flowchart 42
- presentation components (web application tier 1) 195
- presenting data externally 97–103
- presenting information to the user 260
- pretested loop 53
- Pretty Good Privacy (PGP) 292
- preventing user input errors 261
- primary key 108, 109, 149
- principles of visual communication 99–103, 235
- privacy 182, 281, 355–6
 - and the Australia Card 184–5, 186
 - and data collection 183–6, 223
 - and government data retention laws 187
 - and MyGov 185
 - see also Australian Privacy Principles
- Privacy Act 1988* 185, 186, 223, 281, 355
- problem-solving process
 - for cleaning and merging datasets 271
 - digital 3–20, 188
- processes for secure data storage and exchange 290
 - authentication 293–4
 - checksums 294–5
 - CIA triad 290, 291
 - encryption and decryption 291–2
 - hashing 295–6
 - techniques to protect against threats 290
- processing user input 260
- program data 37
- programmed components
 - data exchange 317–21
 - digital solutions 241–5
- programmed solutions
 - ASCII Aliens 75–85
 - Rusty and the Ezibeats 67–75
- programmers 35
 - controlling digital devices 35–6
- programming
 - basic structures of 53–64
 - control structures 53
 - data structures 54–6
 - event listeners 56
 - global vs. local variables 58
- programming development environments, using criteria to evaluate 254–5
- programming environments
 - with data 171–9
 - for data exchange 319–20
- programming languages
 - from human language to machine language 35–6
 - high-level languages 36–7, 52
 - history 52–3
 - used by programmers 37
- programming projects (prototype) 256–75
- project management approaches 250–4
- proofreading 261
- propagation latency 298
- proportion (visual communication) 98–9, 235
- proto-personas 221
- providing feedback to the user 260
- proximity (CARP) 24
- proximity (visual communication) 102, 235
- pseudocode 40
 - conventions 41, 256–7
 - modular and non-modular 43
 - see also algorithms
- Python 52, 196
- quality of ciphers 338
 - modern ciphers 338–46
- quality of service (QoS) guarantee 299–300
- queuing latency 298–9
- readable modular program generation 256
 - pair programming 258
 - pseudocode conventions 256–7
 - removing ambiguity 257–8
- reading

- text files 269
 - and writing binary files 269–70
- real (data type) 111
- recommendations, justifying and prioritising 282
- record (relational database) 106
- rectangle function 50
- redundant data 106, 271
- referential constraints 150
- refining the solution 245–6
 - low-fidelity prototypes 246
 - pitching an idea 246
- relational database management systems 117–19
- relational databases 95, 105, 106–12, 237
 - for data-driven solutions 131–45
 - entity relationship diagrams 133–9
 - evaluate the developed database plan 142–4
 - explore data needs and requirements 132–3
 - normalisation process 139–42
 - field data types 110–11
 - keys 107–8
 - PostgreSQL database management
 - system data types 111–12
 - uniqueness constraints 108–9
 - using SQL 150–9
- relational schemas 148–9
 - features 149
 - implementing 149–50
- relationships (ER diagrams) 133–4
- reliability (databases) 106, 108
- removing ambiguity (algorithms) 310, 320, 321
- removing ambiguity (programming components) 257–8
- REPEAT loop 53
- repetition (algorithm control structure) 37
- repetition (CARP) 24
- repetition (visual communication) 100, 101, 235
- requirements, evaluating 230
- responsive design 23
- responsive interfaces 233
- RESTful services (Representational State Transfer) 304
- retention and disposal of data 186–7
- robot intelligent systems 206–10
- Robot Operating System (ROS) software 207
- robotics solution, data format 237–8
- RSA (Rivest–Shamir–Adleman) algorithm 292
 - asymmetric encryption 347–50
 - simplified example of mathematical processes 348–9
 - simplified sample algorithm 350
- Ruby on Rails 196
- Rusty and the Ezibeats (web application case study)

- exploring the problem 11–14
- hierarchy chart 13
- IPO charts 13
- mind map 12
- user interfaces 29–31
- Rusty and the Ezibeats (web application case study), prototype programmed solution 66–75
 - backup 67
 - evaluation checklist 74–5
 - fonts and icons 68–9
 - generating user interfaces 67
 - CSS 68, 74
 - HTML 68–74
 - home page layout and content 72–4
 - navigation bar 69–72
 - setting up the development environment 67
 - slideshow effects
 - HTML 72–3
 - JavaScript 73
- safety (interfaces) 28, 232
- SAML (Security Assertion Markup Language) 294
- Sam's Breakfast Bar menu 88–9
- scale (visual communication) 98–9, 235
- screen readers 27
- scrolling containers 317
- scrubbing data 271
- Scrum methodology 252, 253
- second normal form (2NF) 139, 141
- secure data storage and exchange, processes for 290–6
- security and protection 115
 - data management solutions 116–17
 - see also* cybersecurity; data security
- SELECT statement (SQL) 156–7, 159
 - calculations and functions 165–7
 - comparison operators 160
 - conditions (the WHERE clause) 157–8
 - Garden Express case study 159–64
 - grouping 167–8
 - INSERT, UPDATE AND DELETE statements 170, 372–3
 - logical operators 159–60
 - ORDER BY clause 169
 - partial matches (LIKE clause) 160–1
 - selecting all fields or using wildcards 157
 - subqueries 168–9
 - table joins 161–4
 - used by an API 373–4
- selecting an appropriate data format 237
 - for an interactive media solution 239
 - for a mobile application 238–9
 - for a robotics solution 237–8
 - for a web application 238
- selection (IF–THEN–ELSE) statements 37, 41, 53

- self-determined criteria, evaluating using 281–2
- sensors 207
- sequence (operation) 37, 41
- serialisation latency 298
- server-side scripting language for web applications 52, 196
- SHA hash algorithm 295
- shape (visual communication) 98, 235
- shared data 353
- shared data exchange, security of 359
- simplified DES (S-DES) cipher 339–40
 - decryption algorithm 344
 - encryption algorithm 341–4
 - key generation algorithm 340–1
 - sample code 345–6
- sliders 317
- slideshow effects 72–4
- smart light bulb decomposed 7
- social benefits of open data 354
- software 2
- software frameworks and libraries 66
- sorting algorithms 54–5
- space (visual communication) 97, 235
- spelling errors 272
- sprint 252, 253
- SQL (Structured Query Language) 150–1
 - Data Definition Language (DDL) 153–5, 272, 370–2
 - Data Manipulation Language (DML) 155–71, 372–4
 - and Database .NET 152
 - development 151
 - features 153
 - grouping 167–8, 365–7
 - mathematical operations 164–7
 - processing results from the database 265–6
 - SELECT statement 156–71, 373–4
 - sending statement to the database 265
 - subqueries 168–9, 367–8
 - table joins 161–4, 365
 - techniques to de-identify and create open data sets 365–8
 - variants on DBMS 118, 152
- SQLite 118, 152
- SSL (Secure Socket Layer) 302
- standards 66
- Standards Australia (SA) 66
- stored procedures 170
- stream ciphers 338–9
- strings 110, 111
- structured data 189–90
- Structured Query Language *see* SQL
- stylesheets 199
- subqueries (SQL) 168–9, 367–8
- summary data 271
- surveys 224–5
- switching latency 298
- symmetric encryption 339, 346, 351
- syntax errors 55

- Syropods (intelligent robot case study) 206, 207, 209
- system boundaries 210, 211
- systems thinking 6–8, 11
 - application to explore digital solutions 210–13
 - data-driven 122–4
 - to analyse use cases 213–14
 - to resolve problems 215
- table creation, in Database .NET visual editor 150
- table joins (SQL) 161–4, 365
- table schemas 273–4
- table structures (HTML) 317
- test case 277, 278
- test scenario 277
- testing the solution 277
 - collecting user feedback 279–80
 - testing functional requirements have been met 277–8
 - useability heuristics 279
- text colour contrast (accessibility) 26–7
- text (data type) 110, 111
- text files 268
 - reading 268–9
- texture (visual communication) 98, 235
- third normal form (3NF) 139, 141–2
- three-layer approach to data 94–6
- three-tier architecture (web applications) 195–6
- time (data type) 111, 112
- time factors (data acquisition) 112–13
- timeliness of delivery 300–1
- TLS (Transport Layer Security) 302, 359
- tone (visual communication) 97–8, 235
- transmitting data 297–301
 - jitter 299
 - latency 298–9
 - quality of service (QoS) guarantee 299–300
 - timeliness of delivery 300–1
- Triple DES cipher 292, 350
 - symmetric decryption example 351
 - symmetric encryption example 351
- Trithemius cipher 328
- tuples 306
- Turing machine 329
- Twofish 292
- Unified Modelling Language (UML) diagrams 211, 213, 229
- unintentional data loss 28
- unique identifiers 109, 182
- uniqueness constraints (relational databases) 108–9
- universe of discourse 90
- UPDATE SQL statements 170, 373
- use cases
 - representation 213
 - to understand user expectations 213–14
- useability heuristics 279
- useability principles (interfaces) 22, 231
 - accessibility 26–8, 231
 - effectiveness 24–5, 232
 - learnability 24–5, 232–3
 - safety 28, 232
 - utility 22–3, 232
- useability success criteria 25
- user credentials 116–17
- user data, validating 261–2
- user expectations, using use cases to understand 213–15
- user experience design 127
- user feedback 260
 - collecting 279–80
- user input
 - processing 260
 - validating 261–2
 - verifying 262
 - waiting for 260
- user input errors, preventing 261
- user interactions
 - coding for 259–62
 - mobile applications and interactive media 204–5
 - web applications 196–8
- user interfaces 22
 - case studies 29–33
 - defining 224–6
 - developing 230–6
 - useability principles 22–9
 - wireframes 29–31, 32
- user-level access 116
- user stories 252, 253–4
- username 116
- users
 - communicating effectively with 259–60
 - presenting information to 260
- UTF-8, use in making ciphers 328
- utility (interfaces) 22–3, 232
- validating user data 261–2
 - preventing user input errors 261
- validating user input 261–2
- value (relational database) 106
- 'varchar' data type 110
- variables 37, 39, 50, 54, 58
- verifying user input 262
- version control 67
- video interviews 226
- Vigenère cipher 329
- visual communication of data 97
 - applying elements and principles 236
 - elements 97–9, 235
 - principles 99–103, 235
- VPN (virtual private network) 303
- VSC (Visual Studio Code) 16
- waiting for user input 260
- waterfall model (project management) 250–1
- web applications
 - coded components 228
 - data format 238
 - elements of 195–201
 - Koala Tracker case study 194–5
 - Rusty and the Ezibeats case study 11–14, 29–31, 66–75
 - three-tier architecture 195–6
- Web Content Accessibility Guidelines (WCAG) 26
- web context, data-driven system components 104, 105
- WHERE conditions 157–8
- WHILE loop 53
- wildcards 157
- wireframes 14, 29–31, 32
- wisdom 90
- World Wide Web consortium (W3C) 26, 66, 296
- XML (Extensible Markup Language) 306–8
 - Digital Signature 293–4
- XMLHttpRequest method 322, 323
- Zoho Sprints, to create user stories 253–4

- Comprehensively covers the new QCAA Digital Solutions Senior Syllabus
- Written by highly experienced Queensland-based Technologies teachers
- A practical, problem-based, learning framework underpins the text with application to real-world scenarios
- Centred around building 21st-century skills of communication, collaboration and teamwork, critical thinking, personal and social skills and ICT skills
- Projects provide opportunities for students to apply their knowledge and skills and prepare for school based and external assessments