Software Engineering Stage 6 (Year 12) – teacher support resource

Software engineering project

# Teacher support resource

**Teacher note:** this resource has been designed to facilitate the ready conversion into a student booklet by removing the answers within the response windows. Teacher notes can be deleted before distributing to students. This booklet should be submitted as the documentation component of the assessment task.

Student name:

Class:

Teacher:

Contents

[Teacher support resource 1](#_Toc165554082)

[Unit overview 5](#_Toc165554083)

[Identifying and defining 5](#_Toc165554084)

[Research, planning and issues 5](#_Toc165554085)

[Producing and implementing 6](#_Toc165554086)

[Testing and evaluating 6](#_Toc165554087)

[Assessment task overview 7](#_Toc165554088)

[Project description 8](#_Toc165554089)

[Submission details 10](#_Toc165554090)

[Steps to success 11](#_Toc165554091)

[Glossary 14](#_Toc165554092)

[NESA glossary keywords 17](#_Toc165554093)

[The Software engineering project 18](#_Toc165554094)

[Personal interest 18](#_Toc165554095)

[Course review and preview 20](#_Toc165554096)

[Developing a project proposal 21](#_Toc165554097)

[The design and production process 23](#_Toc165554098)

[1. Identifying and defining 24](#_Toc165554099)

[Define and analyse problem requirements 24](#_Toc165554100)

[Assess the scheduling and financial feasibility 25](#_Toc165554101)

[Boundaries 26](#_Toc165554102)

[Explore tools to develop ideas and generate solutions 27](#_Toc165554103)

[Investigate types of software implementation methods 28](#_Toc165554104)

[2. Research and planning 30](#_Toc165554105)

[Project management 30](#_Toc165554106)

[Software development approaches 30](#_Toc165554107)

[Apply project management to plan and conduct the development and implementation of a project and software engineering solution 32](#_Toc165554108)

[Using collaboration tools 33](#_Toc165554109)

[Exploring the social and ethical aspects of software engineering projects 33](#_Toc165554110)

[Explore communication issues associated with project work 35](#_Toc165554111)

[Quality assurance 35](#_Toc165554112)

[Ensuring requirements are met using a continual checking process 35](#_Toc165554113)

[Compliance and legislative requirements 35](#_Toc165554114)

[Systems modelling 37](#_Toc165554115)

[Data flow diagrams 39](#_Toc165554116)

[Structure charts 41](#_Toc165554117)

[Class diagrams 43](#_Toc165554118)

[Storyboards 44](#_Toc165554119)

[Decision trees 45](#_Toc165554120)

[Algorithm design 46](#_Toc165554121)

[Explain the contribution of back-end engineering to the success and ease of software development 47](#_Toc165554122)

[3. Producing and implementing 49](#_Toc165554123)

[Design, construct and implement a solution to a software problem using appropriate development approach(es) 49](#_Toc165554124)

[Present a software engineering solution using presentation software 49](#_Toc165554125)

[Develop, construct and document algorithms 49](#_Toc165554126)

[Allocate resources to support the development of a software engineering solution 50](#_Toc165554127)

[Demonstrate the use of programmed data backup 51](#_Toc165554128)

[Implement version control when developing a software engineering solution 52](#_Toc165554129)

[Explore strategies to respond to difficulties when developing a software engineering solution 53](#_Toc165554130)

[Propose an additional innovative solution using a prototype and user interface (UI) design 54](#_Toc165554131)

[4. Testing and evaluating 55](#_Toc165554132)

[Apply methodologies to test and evaluate code 55](#_Toc165554133)

[Use a language-dependent code optimisation technique. 57](#_Toc165554134)

[Analyse and respond to feedback 59](#_Toc165554135)

[Evaluate the effectiveness of a software engineering solution 60](#_Toc165554136)

[Developing a report to synthesise feedback 60](#_Toc165554137)

[Developing a test plan 61](#_Toc165554138)

[Comparing actual output with expected output 61](#_Toc165554139)

[Testing data used/generated based on path and boundary testing 61](#_Toc165554140)

[References 63](#_Toc165554141)

# Unit overview

This unit of work guides students through the development of their Software engineering project. The content for this focus area should be delivered over 30 hours. It should be integrated with other focus areas to support students to apply deeper knowledge, understanding and skills in an area of personal interest. Students develop the knowledge, understanding and skills associated with project development, including identifying and defining requirements, research and planning approaches and issues, producing and implementing software solutions and testing and evaluating code and engineering solutions.

Throughout this sequence of learning, students have opportunities to seek feedback to inform the development of their project. The Software engineering project is developed in the classroom under the supervision of the teacher using explicit teaching methods outlined in this Teacher Support Resource (TSR).

## Identifying and defining

Students develop a project proposal. This includes reflection upon personal interest and their success with projects from Year 11 focus areas (Programming fundamentals, Object-oriented paradigms and Programming mechatronics). Students preview syllabus content for upcoming focus areas (Secure software architecture, Programming for the web and Software automation). They identify, define and analyse the requirements of a problem (or opportunity). Students explore tools used to develop ideas and generate solutions and investigate types of software implementation methods. They contact and interview a client as well as analyse software engineering case studies, scenarios and existing solutions.

## Research, planning and issues

Students research and use the Waterfall, Agile and WAgile software development approaches. They apply project management skills to plan and conduct the development and implementation of their software engineering solution. Students explore social and ethical issues associated with project work, including working individually, collaboratively and responding to stakeholders. They explore communication issues associated with project work and investigate how software engineering solutions are quality assured. Students demonstrate the use of modelling tools and explain the contribution of back-end engineering to the success and ease of software development.

## Producing and implementing

Students construct and implement a solution to a software problem or opportunity using (an) appropriate development approach(es). They construct and document algorithms, demonstrate the use of programmed data backup, implement version control and explore strategies to respond to difficulties when developing their solution. Students propose an additional innovative solution using a prototype and user interface (UI) design.

## Testing and evaluating

Students apply methodologies to test and evaluate code. They use a language-dependent code optimisation technique, analyse and respond to feedback and evaluate the effectiveness of a software engineering solution.

The software engineering solution is evaluated and tested to ensure its features meet the success criteria outlined in the problem definition and identified needs.

Students present their software engineering solution using presentation software to the class (and client) and submit the project documentation and code.

# Assessment task overview

**Type of task**: develop a software engineering project containing a solution, project documentation and a presentation.

**Outcomes being assessed**:

A student:

* justifies methods used to plan, develop and engineer software solutions **SE-12-01**
* applies structural elements to develop programming code **SE-12-02**
* analyses how current hardware, software and emerging technologies influence the development of software engineering solutions **SE-12-03**
* evaluates practices to safely and securely collect, use and store data **SE-12-04**
* explains the social, ethical and legal implications of software engineering on the individual, society and the environment **SE-12-05**
* justifies the selection and use of tools and resources to design, develop, manage and evaluate software **SE-12-06**
* designs, develops and implements safe and secure programming solutions **SE-12-07**
* tests and evaluates language structures to refine code **SE-12-08**
* applies methods to manage and document the development of a software project  
   **SE-12-09**

[Software Engineering 11–12 Syllabus](https://curriculum.nsw.edu.au/syllabuses/enterprise-computing-11-12-2022) © NSW Education Standards Authority (NESA) for and on behalf of the Crown in right of the State of New South Wales, 2022.

**Suggested weighting**: 30%

Students identify a real-world problem or opportunity that can be addressed through the development of a software engineering project. Students develop project documentation and engineer a software solution that addresses this real-world problem or opportunity. The software engineering project is presented to the class by simulating a client handover.

## Project description

****Students find a client for whom they can develop a software solution. Students unable to find a client may use Scenario 1 or Scenario 2 in the Student support material section of the assessment task.

**Possible clients include:**

* school-based positions: a teacher, librarian or school administrator
* a member of the wider community
* business owner
* a non-government organisation (NGO) that works on various humanitarian and social issues
* a sporting club, coach, manager or gym
* a PCYC, YMCA, Scouts, Guides or similar
* a community group, religious group or hobby group
* any other organisation or client approved by the teacher.

**Students meet and negotiate with their client to:**

* define and analyse requirements
* assess scheduling feasibility
* define boundaries
* continually check progress
* document all correspondence.

**Note:** students ensure their classroom teacher is involved in all contact and correspondence between themselves and their clients. This strengthens school partnerships to ensure a quality learning experience for the student and a quality software product for the client. It is the students’ responsibility that both course and client requirements are met. School-defined submission dates and client requirements will assist in determining the scale and scope of the final project.

**The software solution:**

* demonstrates the design, development and construction of algorithms
* addresses all the clients functional and performance requirements.

The solution should also:

* use a language-dependent code optimisation technique
* be configurable in terms of the interface layout, colour scheme, accessibility requirements, legislative and technical requirements as per client needs
* incorporate security elements such as usernames, passwords, basic encryption and decryption as per client requirements
* include a proposal for an additional innovative solution using a prototype and user interface (UI) design.

## Submission details

Students submit:

**Component A – project documentation**

Complete resource booklet including:

* process diary
* client correspondence and feedback
* Gantt chart
* use of modelling tools.

**Component B – software solution**

* Develop, code and upload a software solution for the problem as indicated in the project documentation.

**Component C – presentation**

* Present their software engineering project to peers
* Respond to Q&A.

## Steps to success

Table 1 – assessment preparation schedule

|  |  |
| --- | --- |
| Steps | What I need to do |
| Component A –  Project documentation | Ongoing completion of the Software engineering project documentation booklet provided that involves 4 key stages.   1. Identifying and defining 2. Research and planning 3. Producing and implementing 4. Testing and evaluating |
| Identifying and defining | * Begin by defining and analysing the problem requirements including: * demonstrating need(s) or opportunities * assessing scheduling and financial feasibility * generating requirements including functionality and performance * defining data structures and data types * defining boundaries * Provide ongoing screen shots of tools used to develop ideas and generate solutions including: * brainstorming, mind mapping and storyboards * data dictionaries * algorithm design * code generation * testing and debugging * installation * maintenance. |
| Research and planning | * Apply ongoing project management to plan and conduct the development and implementation of the project including: * scheduling and tracking using a software tool, including Gantt charts * using collaboration tools * Explore communication issues associated with project work including: * involving and empowering the client * enabling feedback * negotiating * Investigate how software engineering solutions are quality assured including: * defining criteria on which quality will be judged * ensuring requirements are met using a continual checking process * addressing compliance and legislative requirements * Demonstrate the use of modelling tools including: * data flow diagrams * structure charts * class diagrams * decision trees. |
| Component B –complete software solution | * Develop and code a software solution for the problem as indicated in the project documentation * Upload the solution. |
| Producing and implementing | * Design, construct and implement a solution to a software problem using appropriate development approach(es) * Develop, construct and document algorithms * Implement version control when developing a software engineering solution * Propose an additional innovative solution using a prototype and user interface (UI) design. |
| Testing and evaluating | * Apply methodologies to test and evaluate code * Use a language-dependent code optimisation technique. |
| Component C – presentation | * Present their software engineering project to peers * Respond to questions and answers. |
| Testing and evaluating | * Analyse and respond to feedback * Evaluate the effectiveness of a software engineering solution. |

## Glossary

Many of the following words will gather more meaning to you as you work through this booklet. Each time you see a new word in bold throughout this workbook you can add its definition in the table below in case you need to refer back later.

Table 2 – glossary

|  |  |
| --- | --- |
| Word | Definition |
| agile | An iterative and flexible approach to software development, focusing on collaboration, customer feedback and small, rapid releases. |
| algorithm | A step-by-step procedure required to solve a problem. Algorithms may be presented in many ways, for example written instructions, flow charts or using a computer programming language. |
| app | A software program designed for a specific purpose to run on mobile devices or on a personal computer. An abbreviation of the word ‘application’. |
| Application Programming Interface (API) | An interface that allows an application or website to plug into another program or website. |
| boundary | An invisible line that separates the software system being built from everything else. It helps define the limits and expectations of a software system. |
| client | The person or organisation who requires the solution and is likely to directly use the software application. A person or organisation using the services of a software engineer. |
| code | The instructions that guide a program’s execution. |
| code optimisation | To modify existing code or design new algorithms to improve program execution speed, reduce memory usage and enhance overall system performance |
| computational thinking | A process in which a problem is analysed and solved so that a human, machine or computer can effectively implement the solution. It involves using strategies to organise data logically, break down problems into parts, interpret patterns and design and implement algorithms to solve problems. |
| data flow diagrams | Diagrams illustrating the flow of data within a system, showing how it is processed or transformed. |
| data structures | A data structure is a way of organising and storing data in a computer so that it can be accessed and used efficiently. |
| data types | Data types used in computing are expressed as either: string or text, character, integer, floating point or real, date and time and Boolean. |
| decision trees | Diagrams that represent decisions and their possible consequences, often used in decision-making processes. |
| end-user | The person or group who will ultimately use the system or product. |
| Gantt charts | Visual tools used for project management to represent the timing of tasks or activities. |
| iterative approach | A method of development where the project is divided into smaller parts and developed incrementally, with each iteration building upon the previous one. |
| Object-Oriented Programming (OOP) | A paradigm based on the concept of ‘objects’ that can contain data and code in the form of procedures. |
| Object-Relational Mapping (ORM) | A technique used in object-oriented programming to query and manipulate data from a database. |
| online collaboration | Working together on projects or tasks using internet-based tools and platforms. |
| outsourcing | Hiring external individuals or companies to perform tasks or services instead of internal employees. |
| procedural programming | A method of programming where the program is divided into functions. A program consists of data and procedures (modules) that operate on the data. Data and procedures are treated as separate entities. |
| prototyping | Building partial versions of a system to test ideas and gather feedback before full implementation. |
| pseudocode | A form of algorithm description that uses English-like statements with defined rules of structure and keywords. |
| system flowcharts | Visual representations of the flow of data or processes within a system. |
| verification and validation | Processes to ensure that the system meets specified requirements and functions correctly. |
| WAgile | WAgile is a hybrid of Waterfall and Agile Project Management |
| waterfall (structured) | A sequential development approach where progress flows steadily downward through predefined phases. |
| working remotely | Performing work from a location other than the traditional office setting, often enabled by technology and the internet. |

**Teacher note:** for students with an EAL/D background. The glossary can be provided complete so that they have additional time to understand the key terms with bilingual dictionaries. The glossary can be provided to students in their preferred communication mode.

## NESA glossary keywords

NESA keywords can be used in the syllabus and in the Higher School Certificate examination. Familiarisation with these keywords can assist in understanding how to write and respond to questions.

|  |  |
| --- | --- |
| Key term | Definition |
| ****Apply**** | Use, utilise, employ in a particular situation. |
| ****Describe**** | Provide characteristics and features. |
| ****Explain**** | Relate cause and effect; make the relationships between things evident; provide why and/or how. |
| ****Investigate**** | Plan, inquire into and draw conclusions about. |

[Glossary of key words](https://www.nsw.gov.au/education-and-training/nesa/hsc/student-guide/glossary) © NSW Education Standards Authority (NESA) for and on behalf of the Crown in right of the State of New South Wales, 2024.

**Teacher note:** develop, explore, select and verify are used in this topic and are not listed.

# The Software engineering project

Teachers assist students in identifying a project that is engaging, enjoyable and achievable. The project should sustain interest for the duration of the Year 12 course and challenge student capability to outside of their ‘comfort zone’. Student interest, real-world problems and opportunities with actual clients have proven to be powerful motivators for success. They are encouraged to collaborate with critical friends and to interview clients with questions around how a digital solution may assist to reduce frustrations or add value to their work and/or leisure lives.

## Personal interest

[Think-Pair-Share](https://app.education.nsw.gov.au/digital-learning-selector/LearningActivity/Card/645): list personal interests or commitments that may inspire your software engineering project in the table below. Pair up with a classmate to discuss your top 3 interests and describe some of the processes involved in these interests.

|  |  |
| --- | --- |
| Interest/commitment (Sample answers) | Description (Sample description) |
| Pets | Training, monitoring and care. |
| Family and friends | Communicating, diarising. |
| Health, exercise, and nutrition | Monitoring health and diet. |
| Hobby | Gaming, hosting. |
| Sport | Registering and performance. |
| Study | Scheduling, revision and testing. |
| Work | Digitising, automation, workflow. |
| Travel | Planning, budgeting and itineraries. |
| Shopping | Searching specials. |
| Transport | Comparing energy efficiency. |

After you have discussed your top 3 interests with your partner, complete the following steps.

1. Share with the class.
2. Brainstorm possible clients for your software engineering project.
3. Begin researching existing solutions.

## Course review and preview

Determine the scale and scope of the Software engineering project and your capacity to achieve it by considering what you have learnt in the Year 11 course and what you’ll be learning in Year 12.

Reflect upon projects from the Year 11 course in the table below.

|  |  |
| --- | --- |
| Focus area | Projects |
| Programming fundamentals |  |
| Object-oriented paradigms |  |
| Programming mechatronics |  |

Which of these were most engaging? Explain why in the space provided below.

|  |
| --- |
|  |

Preview the [syllabus content](https://curriculum.nsw.edu.au/learning-areas/tas/software-engineering-11-12-2022/overview) for upcoming focus areas listed in the table below.

|  |  |
| --- | --- |
| Focus area | Description |
| Secure software architecture |  |
| Programming for the web |  |
| Software automation |  |

Which of these topics integrate into the Software engineering project? Explain how in the space provided below.

|  |
| --- |
|  |

## Developing a project proposal

Students [create a mind map](https://www.mindmaps.com/project-planning-with-mind-maps/) using a [Lucidchart](https://www.lucidchart.com/pages/?) of their software engineering project proposal by linking the key concepts and findings from their:

* possible clients list
* personal interests
* Year 11 focus area projects review
* Year 12 focus areas preview.

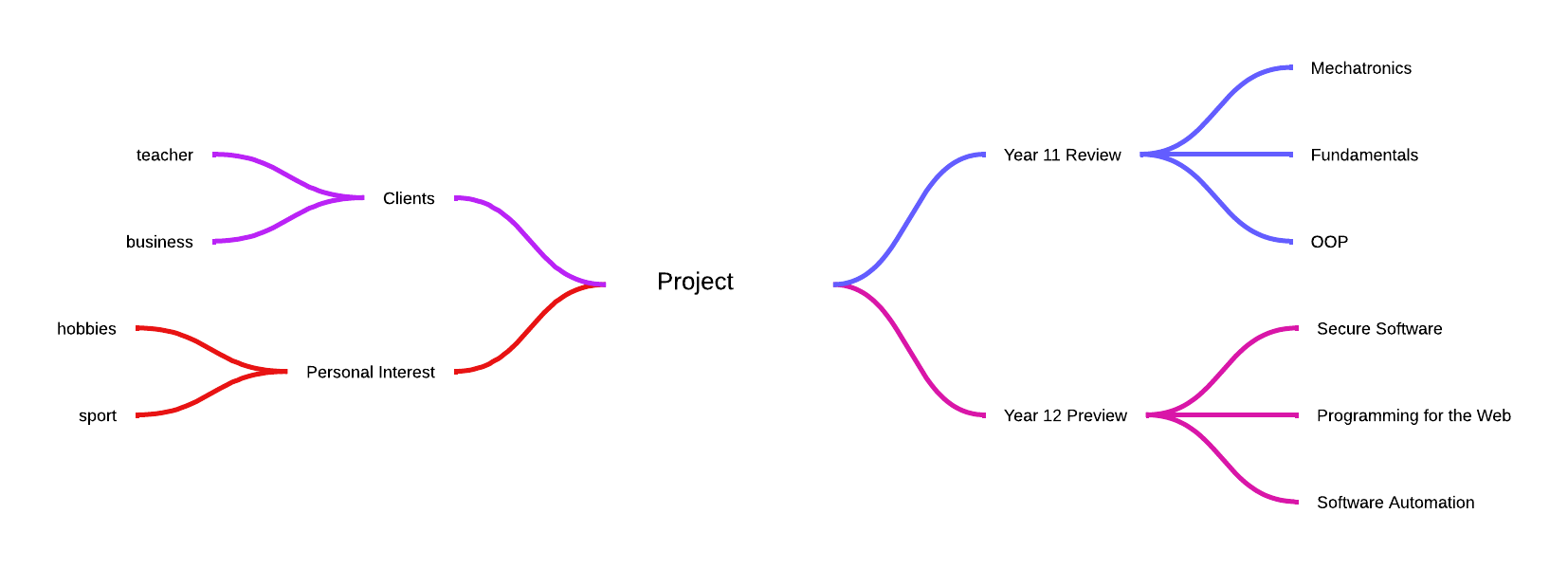
The development of a mind map will assist students to identify and find connections between:

* what they plan to make
* who they will make it for
* how they will make it.

Figure 1 shows an incomplete mind map with 4 key aspects of the software engineering project: possible clients, personal interests, Year 11 focus areas and Year 12 focus areas.

1. Extend each branch of the mind map as far as possible into individual ‘leaves’.
2. Circle the ‘leaves’ that inspire or resonate most strongly.
3. Find connections between ‘leaves’ from other branches.
4. Annotate these connections into possible project ideas.
5. Rank these ideas into a list of project preferences ensuring that a backup plan is identified.
6. Share your list with a critical friend, peer and teacher.

Figure 1 – incomplete mind map using Lucidchart



## The design and production process

Throughout your study of Software engineering, you will learn about different types of design processes and how to apply them in your design project.

The design and production process:

* involves a sequence of organised steps which provide a solution to design needs and opportunities
* may take a few seconds or minutes, such as when you select what clothes to wear, or may take years as in the case with the design of a motor vehicle
* may involve one person or may involve many people
* may be simple or complex, depending on the task
* involves questioning (or evaluating) throughout the iterative process.

Figure 2 – flowchart of design and production process

Design and production process diagram
A flowchart labelled 'Ongoing evaluation' with a two-headed arrow indicating both directions. 
The first part of the flowchart is called '1. Identifying and defining'. It says 'identify and define the needs, opportunities and wants of a computing challenge, practise the technical skills, develop evaluation criteria.' There is an arrow pointing to the next section, which is labelled '2. researching and planning'. It says 'research, generate and practise ideas, be creative and propose new approaches to problems, explore new design opportunities.' An arrow points to the next section, labelled '3. producing and implementing', it says 'build and implement ideas, apply a variety of skills and techniques to create products that meet set criteria, modify and iterate solutions'. The arrow points to the next section, labelled '4. testing and evaluating'. It says 'test and evaluate solutions/products, evaluate quality and effectiveness against the criteria, make judgements throughout the solution and use these to refine the product.'
After testing and evaluating is a big arrow called 'Review if required to improve' and it goes all the way back up to the first part of the flowchart, indicating a cycle.

# 1. Identifying and defining

## Define and analyse problem requirements

Discuss with the client their needs and propose further opportunities. These will include the functional and performance requirements. In the table below describe the need(s) and opportunities that the software solution will provide for the client.

|  |  |
| --- | --- |
| Need | Opportunities |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

Rank the needs and opportunities into priorities(‘must haves’) and opportunities (‘like to have’) by numbering the list.

**Teacher note:** this project is due at the end of Term 3.

Will students have the time and money (if required) to complete the project for both the client and school submission?

### Assess the scheduling and financial feasibility

* Which of the needs and opportunities will be achieved first?
* Which are dependent upon the previous completion of another?
* Will there be any cost in this project?

In consultation with the client generate a ‘requirements of the problem’ statement that includes functionality (*Does it do what it is meant to do?*) and performance (*How well does it do it?*).

|  |
| --- |
|  |

Revise the use of data structures and datatypes from the Year 11 projects.

In consultation with the client, analyse the problem. The data structures and data types required will need to be defined. These are presented in data dictionaries when modelling the system.

List all of the entities (things) that will be represented and stored digitally in your software engineering solution and the data structures and data types that will be used to do so. For example:

* customers could be represented as an array of records or a database.
* high scores as integer variables.
* monster as an object from the character class.

### Boundaries

Boundaries define the limits of the problem or the system to be developed.

Defining the boundaries for the problem is essential so the client has realistic expectations of the limits of the new system.

Consider the software engineering solution and the aspects you can control.

Which parts of the problem remain outside the system (in the environment) that will interact with the system through an interface?

In the space below discuss any limitations or boundaries in which this new software will need to operate.

|  |
| --- |
|  |

## Explore tools to develop ideas and generate solutions

The following tools will gather meaning as you work through this booklet. Each time you use one of these tools take a screenshot and paste it into this workbook. Add a description of its use in the table below.

|  |  |  |
| --- | --- | --- |
| Tools | Screenshot | Description |
| Algorithm design |  |  |
| Brainstorming |  |  |
| Code generation |  |  |
| Data dictionaries |  |  |
| Debugging |  |  |
| Installation |  |  |
| Maintenance |  |  |
| Mind-mapping |  |  |
| Storyboards |  |  |
| Testing |  |  |

## Investigate types of software implementation methods

****Software implementation methods are:

* direct
* phased
* parallel
* pilot.

Each of these implementation methods has its advantages and disadvantages, and the choice depends on the specific context of the project, including its size, complexity and the organisation's risk tolerance.

**Direct implementation** involves switching from the old system to the new system in one single action. All users move to the new system on a set date, and the old system is retired immediately. This method is straightforward and quick but can be risky if the new system has not been thoroughly tested.

**Parallel implementation** means both the old and new systems run simultaneously for a period. This allows for a comparison between the 2 systems to ensure that the new system operates correctly before the old system is decommissioned. While safer than direct implementation, running 2 systems in parallel can be resource intensive.

**Pilot implementation** involves rolling out the new system to a small, manageable group of users before a full-scale implementation. This method allows organisations to identify any issues or necessary adjustments in a controlled environment, reducing the risk of widespread problems.

**Phased implementation** involves gradually implementing the new system in phases or modules. Each phase is rolled out and stabilised before moving on to the next, allowing for incremental testing, training and adaptation. This method can reduce risk and disruption but may take longer to fully implement the new system.

Figure 3 – the 4 implementation methods.



In the space below **explain** the applicability of the implementation method for the current project.

|  |
| --- |
|  |

# 2. Research and planning

## Project management

### Software development approaches

[**Jigsaw activity**](https://app.education.nsw.gov.au/digital-learning-selector/)

Jigsaw is a cooperative learning strategy that:

* supports educators to differentiate learning
* enables each student to specialise in one aspect of a topic, with each student's part being essential to the completion of the task
* builds students' comprehension, cooperation, communication and problem-solving skills.

1. Assign students into ’home groups’, with 3 students in each home group.
2. Students in each home group are numbered 1, 2, 3.
3. All students in home group 1 form an expert group that will research Waterfall.
4. All students in home group 2 form an expert group that will research WAgile.
5. All students in home group 3 form an expert group that will research Agile.
6. Each group completes the research task below in the tables below and returns to the home group to teach the other group members about the software development approach they have researched.

**Expert group 1 – the Waterfall software development approach**

|  |  |
| --- | --- |
| Question | Sample explanation |
| 1.1 How are the logical progression of steps used throughout the life cycle? | [Insert sample explanation] |
| 1.2 What are the stages of ‘falling water’? |  |
| 1.3 What are the advantages and disadvantages of this approach. |  |
| 1.4 Give examples of the scale and types of developments that use this approach. |  |

**Expert group 2 – the WAgile software development approach**

|  |  |
| --- | --- |
| Question | Sample explanation |
| 2.1 Explain why it is a hybrid model | [Insert sample explanation] |
| 2.2 Analyse the ‘when’ intervention is applied during the development life cycle |  |
| 2.3 Analyse the ‘how’ intervention is applied during the development life cycle |  |
| 2.4 Give examples scale and types of developments that use this approach |  |

**Expert group 3 – the Agile software development approach**

|  |  |
| --- | --- |
| Question | Sample explanation |
| 3.1 What is the rate of developing a final solution? | [Insert sample explanation] |
| 3.2 Explain method tailoring |  |
| 3.3 Explain iteration workflow |  |
| 3.4 Give examples of the scale and types of developments that use this approach |  |

1. Expert group members return to their home groups and peer instructthe other members of the home group on their research findings.
2. Each home group presents to class:

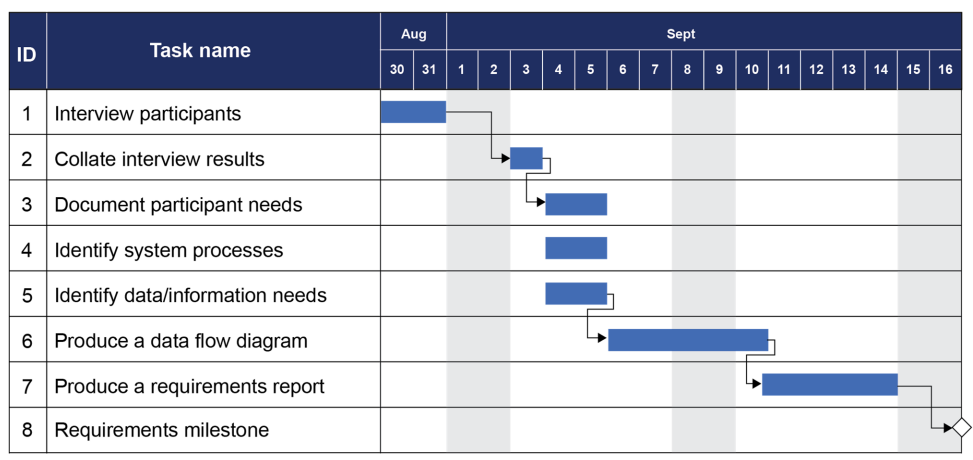
* a Venn diagram that shows the 3 Software development approaches similarities and differences.
* a justification of the use of one of these approaches for their current software engineering project.

### Apply project management to plan and conduct the development and implementation of a project and software engineering solution

#### Scheduling and task allocation

Students refer to the [Software Engineering HSC Course Specifications (PDF 2.9 MB)](https://library.curriculum.nsw.edu.au/341419dc-8ec2-0289-7225-6db7f2d751ef/94e1eb0a-0df7-4dbe-9b72-5d5e0d17143a/software-engineering-11-12-higher-school-certificate-course-specifications.PDF) to develop a Gantt chart that details the tasks required to be completed, person or people assigned to each task, timeline that does not exceed the project due date and resources required. A sample Gantt chart is shown in the figure below.

Figure 4 – sample Gantt chart



Gantt chart from Higher School Certificate Course Specifications – Software Engineering (NESA 2023).

### Using collaboration tools

In the table below **identify** and describe collaborative tools used to manage this project and develop the solution, for example: Repl.it, GitHub, and so on.

|  |  |  |
| --- | --- | --- |
| Name | Description | Screenshot |
|  |  |  |
|  |  |  |
|  |  |  |

### Exploring the social and ethical aspects of software engineering projects

Consideration of social and ethical issues enables the development of software that not only meets technical requirements but also has a positive impact on society.

Complete a description for each key issue listed below. A sample answer has been provided in the table below.

|  |  |
| --- | --- |
| * Privacy | * Fairness |
| * Security | * Intellectual Property |
| * Accessibility and inclusivity | * Collaboration |
| * Transparency | * Feedback |

|  |  |
| --- | --- |
| Social and ethical aspects | Description |
| Privacy | Data collected and processed by software must be done so with the user's consent and comply with relevant regulations. |
| Security | Software must be developed with robust security measures to protect against cyber threats and data breaches. |
| Accessibility and inclusivity | Software must be accessible to all users, including being inclusive of those with disabilities, by following accessibility guidelines. |
| Transparency | Transparency around how software works, including any algorithms or AI systems used, builds trust with stakeholders. |
| Fairness | Software must not discriminate against any group or individual, and consideration needs to be given to the potential biases in algorithms. |
| Intellectual Property | Intellectual property rights must be respected including copyrights, patents and trademarks, when developing software. |
| Collaboration | Working collaboratively with team members and stakeholders ensure that the software meets their needs and addresses any concerns. |
| Feedback | Being open to feedback from stakeholders and being willing to make changes to the software based on their input ensures better results. |

### Explore communication issues associated with project work

As a class role play the importance of communication issues during project work by developing short acts that demonstrate both the correct and incorrect ways to do the following:

* involve and empower the client
* enable feedback
* negotiate.

### Quality assurance

**Quality criteria**

Students **explain** quality criteria based upon the needs and functional requirements. These quality criteria should contain qualities, characteristics or components that need to be included or visible by the end of the current project.

Table 3 – criteria and explanation

|  |  |
| --- | --- |
| Quality criteria | Explanation |
|  |  |
|  |  |
|  |  |

### Ensuring requirements are met using a continual checking process

Students maintain Gantt or scheduling charts to indicate where continual checking process are ensuring requirements are met. Minutes of meetings, emails and diarised notes should be retained as evidence that the projects are on track. These can be kept in the log.

Compliance and legislative requirements

Students **explain** compliance and legislative requirements their projects need to meet and how they plan to mitigate them where possible. For example, projects that deal with sensitive personal data being publicly available may fall under the [Privacy and Personal Information Protection Act 1998](https://legislation.nsw.gov.au/view/whole/html/inforce/current/act-1998-133" \l "statusinformation) (NSW) and/or [*Privacy Act 1988*](https://www.legislation.gov.au/Series/C2004A03712) (Cth). Alternatively, international standards on information security management such as [ISO/IEC 27001](https://www.iso.org/standard/27001) may also be applicable.

Table 4 – compliance and legislation

|  |  |
| --- | --- |
| Compliance or legislative issue | Methods for mitigation |
|  |  |
|  |  |
|  |  |

### 

## Systems modelling

Students are to **develop** the given tables and diagrams. Students should consult the [Software Engineering HSC Course Specifications (PDF 2.9 MB)](https://curriculum.nsw.edu.au/learning-areas/tas/software-engineering-11-12-2022/overview#software-engineering-course-specifications-software_engineering_11_12_2022) guide should they require further detail, exemplars or information. Each subsection below should be completed with reference to the functional requirements.

**Data dictionaries and data types**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Variable | Data type | Format for display | Size in bytes | Size for display | Description | Example | Validation |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Variable | Data type | Format for display | Size in bytes | Size for display | Description | Example | Validation |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |

### Data flow diagrams

Data flow diagrams (DFDs) are used in software engineering to visualise the flow of data within a system. They are a graphical representation that shows how data is input into the system, processed and output. Complete the table below on the uses of Data flow diagrams.

|  |  |
| --- | --- |
| Uses of data flow diagrams | Description |
| Understand system requirements | DFDs help stakeholders, including developers, clients and users understand the flow of data within the system. They provide a clear and visual representation of how data moves through the system, which helps understanding system requirements. |
| Identify data sources and destinations | DFDs identify the sources of data input into the system (for example, users, external systems) and the destinations of data output (for example, databases, reports). This helps in designing interfaces and data storage structures. |
| Define processes | DFDs define the processes or functions that transform input data into output data. Each process in a DFD represents a specific action or operation that the system performs on the data |
| Show data stores | DFDs show where data is stored within the system. This includes both temporary storage (for example, variables and buffers) and permanent storage (for example, databases and files). |
| Identify data flows | DFDs show the paths along which data flows within the system. This includes both the flow of data between processes and the flow of data between processes and data stores. |
| Analyse system behaviour | DFDs can be used to analyse the behaviour of the system and identify potential bottlenecks, inefficiencies or areas for improvement. By visualizing the data flow, developers can optimize the system's performance and efficiency. |
| Document system design | DFDs serve as a documentation tool for system design. They provide a visual representation of the system's architecture, which can be used to communicate design decisions to stakeholders and developers. |

As a class read [What is a data flow diagram?](https://miro.com/diagramming/what-is-a-data-flow-diagram/)

|  |  |  |
| --- | --- | --- |
| Symbol name | Symbol | Description |
| Processes | Process symbol. | These circles show how data is processed within the system.  A circle represents a process. A process uses input(s) to generate output(s). |
| Data stores | Data store symbol. | Depicted as open-ended rectangles, they indicate where data is stored.  A data store can be an electronic file or non-computer storage. |
| External entity | External entity symbol. | Illustrated by squares, these are sources or destinations of data outside the system.  An external entity can be any person, organisation or element that provides data to the system or receives data from the system. |
| Data flow | **Data flow symbol.** | A labelled, curved arrow represents the flow of data between processes, data stores and external entities. |

#### DFD levels

**Level 0 (Context diagram) –** this provides a high-level overview of the system, showing the system as a whole and its interactions with external entities.

**Level 1 DFD** **–** offers more detail than the context diagram by breaking down the system into major processes, showing how data flows between them and external entities. It provides a more granular look at the system's operation but keeps the focus on the overall system rather than minute details.

**Level 2 DFD and beyond –** these diagrams dive deeper into each process depicted in a Level 1 DFD, detailing the sub-processes and their data flows. The further you go beyond Level 2, the more detailed and focused the examination of processes and data flows becomes.

When creating a DFD, start with identifying major inputs and outputs, then build a context diagram (Level 0). Expand this into a Level 1 DFD by detailing major processes and how data flows between them. You can further detail these processes in a Level 2 DFD if necessary. Always ensure your diagram is accurate and easily understandable, checking with others for comprehension.

Examine examples of Level 0 and Level 1 DFDs in the [Software Engineering HSC Course Specifications (PDF 2.9 MB)](https://library.curriculum.nsw.edu.au/341419dc-8ec2-0289-7225-6db7f2d751ef/94e1eb0a-0df7-4dbe-9b72-5d5e0d17143a/software-engineering-11-12-higher-school-certificate-course-specifications.PDF).

Use specialised software to create [Data Flow Diagrams](https://miro.com/diagramming/data-flow-diagram/) for the Software engineering project.

### Structure charts

Teachers explicitly teach the development of a structure chart that is familiar to students. Teachers demonstrate how to construct a structure chart to represent a library system. An example can be found in the [Software Engineering HSC Course Specifications (PDF 2.9 MB)](https://curriculum.nsw.edu.au/learning-areas/tas/software-engineering-11-12-2022/overview#software-engineering-course-specifications-software_engineering_11_12_2022).

****Constructing a structure chart to represent a library system involves identifying the main modules or components of the system and how they interact with each other.

#### Steps to create a structure chart

1. Brainstorm all the functions that need to be coded. This involves:
2. analysing the problem to determine what is needed
3. using a top-down approach to decompose (break) the problem into the sub functions
4. indenting can be used to show sub functions.
5. Construct the diagram. This involves:
6. drawing a box-rectangle for each function. Use verbs, for example, borrow, return, locate, check, update in each rectangle
7. drawing the interconnecting lines between the boxes, showing how the functions are linked to each other
8. adding symbols to the diagram
9. loops (for example for when a function is called multiple times)
10. case (diamond) – (choice of functions)
11. passing of parameters (open – variable, closed – control).
12. Provide explanations and further details to explain your diagram.

This could include providing a short description of each function and explanations of other aspects of your design that someone else reading your diagram would need to understand your design.

Use the following checklist when making a structure chart. Have you:

1. listed all the ‘sub functions’ for the system?
2. interconnected them correctly to relate the correct relationship between subs?
3. used the correct symbols to relate passing variables, loops and decisions?

Students **develop** a structure chart demonstrating how the procedures, modules or components of their final software solution are interconnected.

### Class diagrams

Class diagrams provide a visual representation of systems that are implemented using the object-oriented paradigm. They model classes, their attributes and methods and the relationships between classes. Teachers explicitly teach the development of a class diagram that is familiar to students. Teachers demonstrate how to construct a structure chart to represent:

* a person
* a student and parent, as well as a subject.

Each diagram will include attributes and methods. Students should create a teacher class that inherits from a person. An example can be found in the [Software Engineering HSC Course Specifications (PDF 2.9 MB)](https://curriculum.nsw.edu.au/learning-areas/tas/software-engineering-11-12-2022/overview#software-engineering-course-specifications-software_engineering_11_12_2022).

****[Think-Pair-Share:](https://app.education.nsw.gov.au/digital-learning-selector/) students consider whether their software engineering project is aproblem or opportunity that can be modelled using real-world entities (things) such as objects, classes, and relationships. Object-oriented Programming (OOP) allows you to create a model that closely mirrors the real-world problem, making it easier to understand and think about. Problems that involve complex systems with multiple interacting components and entities are well-suited for OOP.

Students identify and list the entities (things) in their project. They identify attributes and methods of entities and their relationship to other entities (things). They discuss these with their partner to **develop** a class diagram in the space below that demonstrates how each class is related to the other.

|  |
| --- |
|  |

These are shared in a class discussion and adjustments are made according to feedback. A final example is provided in the documentation for this project.

### Storyboards

A storyboard shows the various interfaces (screens) as well as the links between them. For example the [Software Engineering HSC Course Specifications (PDF 2.9 MB)](https://curriculum.nsw.edu.au/learning-areas/tas/software-engineering-11-12-2022/overview#software-engineering-course-specifications-software_engineering_11_12_2022) storyboard shows the relationship between 3 pages of information aimed at promoting a school canteen on a website.

****Teachers demonstrate the reverse engineering or backward mapping of an existing website or app by suggesting the storyboard that was used to design their interfaces.Students **develop a** storyboard in the space below, with at least 3 screens that visually represent the software solutions they will build.

|  |
| --- |
|  |

### Decision trees

Teachers explicitly teach the development of a decision tree by providing a worked example that is familiar to students. Teachers demonstrate how to construct a decision tree to show the rules that control the temperature system within a ‘smart’ house. Students revisit the Year 11 mechatronics focus area projects to investigate which sensor and actuator behaviours could be developed as decision trees.

Teacher introduces other vertical representations used to decide upon whether or not to buy a car.

Examples can be found in the [Software Engineering HSC Course Specifications (PDF 2.9 MB).](https://curriculum.nsw.edu.au/learning-areas/tas/software-engineering-11-12-2022/overview#software-engineering-course-specifications-software_engineering_11_12_2022)

Students **develop** decision trees to visually outline the logic flow and chain of decisions or selections the final solution will need. Students use the space below to develop a decision tree for the software engineering project.

|  |
| --- |
|  |

### Algorithm design

Software engineering students must be able to develop and interpret algorithms represented as pseudocode and flowcharts. Pseudocode is a text-based method of describing the logic in an algorithm. It makes use of capitalised keywords and indentation to show control structures used. Flowcharts are graphic based diagrams that represent algorithms and are read from top to bottom and left to right. Students should regularly practise making pseudocode from flowcharts and flowcharts from pseudocode.

****In designing complex algorithms, it is essential that students start with a clear, uncluttered mainline. The mainline should reference required subroutines, the details of which are shown in separate algorithms. Each subroutine should be concise and correctly make use of further subroutines for detailed logic.

All algorithms designed should adhere to the [Software Engineering HSC Course Specifications (PDF 2.9 MB).](https://curriculum.nsw.edu.au/learning-areas/tas/software-engineering-11-12-2022/overview#software-engineering-course-specifications-software_engineering_11_12_2022)

Using the space below, students **develop** algorithms using methods such as pseudocode and flowcharts to solve the problem and meet the needs outlined in the Identifying and defining phase of the Software engineering project. These algorithms should explicitly include the variables from the data dictionaries created in the previous section.

|  |
| --- |
|  |

### Explain the contribution of back-end engineering to the success and ease of software development

Backend engineering plays a crucial role in the success and ease of software development by providing the foundation and infrastructure that allows applications to function effectively and securely. Complete the table below to explain how each aspect of backend engineering contributes to the success and ease of software development.

**Homework extension:** watch [All You Need To Know About Backend Engineering (15:13)](https://youtu.be/cLK5vqPv6JQ?si=oiggErEI4rYmItlw).

Complete the definitions on the following page.

|  |  |
| --- | --- |
| Terminology | Description |
| technology used | Backend engineering involves selecting and implementing the appropriate technologies and frameworks to support the application's functionality and scalability.  This includes databases (for example, MySQL and MongoDB), server-side languages (for example, Node.js, Python and Java), and frameworks (for example, Express.js, Django and Spring). |
| error handling | Backend engineers implement robust error-handling mechanisms to ensure that the application can gracefully handle unexpected situations, such as server errors, database failures or input validation issues.  Proper error handling improves the reliability and user experience of the application. |
| interfacing with front end | Backend engineers design APIs (Application Programming Interfaces) that allow the frontend of the application to communicate with the backend. APIs define the protocols and rules for how different software components should interact, enabling seamless integration between the frontend and backend. |
| security engineering | Security is a critical aspect of backend engineering. Backend engineers implement security measures to protect the application from vulnerabilities such as SQL injection, cross-site scripting (XSS), and cross-site request forgery (CSRF). This includes implementing authentication, authorisation, encryption and other security best practices. |

# 3. Producing and implementing

Students present their software engineering project to the class (simulating a handover submission to their client). During this class presentation students demonstrate and explain each of the following.

## Design, construct and implement a solution to a software problem using appropriate development approach(es)

During this process students capture and include screenshots to document the development of their solution.

Each screenshot should include a caption that **explains** how it links to the:

* Component A of the assessment task
* Project Management development approaches

These screenshots are to be included in Component C of the assessment task – the class presentation.

## Present a software engineering solution using presentation software

Students choose presentation software for example, Microsoft PowerPoint, Google Slides, Sway, Prezi, Canva and so forth, to present the screenshots and captions.

## Develop, construct and document algorithms

Students develop, construct and document algorithms used in the software engineering project. They refer explicitly to the [Software Engineering HSC Course Specifications (PDF 2.9 MB)](https://curriculum.nsw.edu.au/learning-areas/tas/software-engineering-11-12-2022/overview#software-engineering-course-specifications-software_engineering_11_12_2022) paper to do so. During the presentation of their software solution they refer to the algorithms developed to write their code.

## Allocate resources to support the development of a software engineering solution

During the development of a software engineering solution a careful allocation of resources increases the likelihood of a successful product. Complete the table below:

|  |  |
| --- | --- |
| Operation | Description |
| Define project scope | Clearly define the goals, objectives and scope of the project. Understand the requirements and constraints to determine the resources needed. |
| Identify required resources | Identify the resources needed for the project, including personnel, technology, infrastructure and budget. |
| Allocate personnel | Assign skilled team members to the project based on their expertise and availability. Consider factors such as experience, knowledge and availability when allocating personnel. |
| Allocate technology | Choose the appropriate technologies and tools needed for the project based on the requirements and objectives. Ensure that the technology is compatible with the project scope and team's skillset. |
| Allocate infrastructure | Determine the infrastructure requirements, such as hardware, software and network resources needed to support the development and deployment of the software solution. |
| Allocate budget | Estimate the budget required for the project including costs for personnel, technology, infrastructure and other expenses. Allocate the budget based on the priorities and critical needs of the project. |
| Monitor and adjust | Continuously monitor the progress of the project and the utilisation of resources. Adjust the allocation of resources as needed to ensure that the project stays on track and meets its objectives. |
| Manage risks | Identify and manage risks that may affect the allocation of resources. Have contingency plans in place to address potential risks and mitigate their impact on the project. |

Highlight the key steps that most relevant to your project.

## Demonstrate the use of programmed data backup

While developing solutions software engineers design and implement a backup strategy. Complete the following table to outline the steps in formulating a programmed data backup.

|  |  |
| --- | --- |
| Operation | Description |
| Identify backup requirements | Determine the data that needs to be backed up: databases, files, configurations, and so on. |
| Determine the backup frequency | What is the retention policy, and what are the backup storage requirements? |
| Select backup tools | Choose backup tools and technologies that meet your requirements. These may include built-in database backup utilities, file backup software or cloud backup services. |
| Implement backup strategy | Develop and implement a backup strategy based on your requirements. This may include scheduling automated backups, specifying backup locations and configuring backup settings. |
| Test backup process | Test the backup process to ensure that it works as expected, that backups are created and can be restored when needed. |
| Monitor backup performance | Monitor the backup process to ensure that backups are being created and stored correctly. Monitor backup performance and address any issues that arise. |
| Demonstrate backup and restore | Explain how backups are scheduled, created, and stored. Then, demonstrate how a backup can be restored to recover lost or corrupted data. |
| Document backup procedures | Document the backup procedures and include them in your software documentation. This helps users understand how backups are managed and how they can be restored if needed. |

Highlight the key steps of the programmed data backup that are most relevant to your project.

## Implement version control when developing a software engineering solution

Implementing version control is crucial when developing a software engineering solution to manage changes to the code, track progress and collaborate effectively.

Research version control systems for software developers and complete the table below.

|  |  |
| --- | --- |
| Version control systems (VCS) for software developers | Description |
| ****Git****: | Git is a popular and widely used distributed version control systems. It is free, open-source, and provides features for tracking changes, branching, and merging. Many Python projects, libraries, and frameworks are hosted on platforms like GitHub, GitLab, and Bitbucket. |

In the space provided describe how you managed version control.

|  |
| --- |
|  |

## Explore strategies to respond to difficulties when developing a software engineering solution

In the space provided below **describe** strategies you explored to respond to difficulties when developing your solution including:

* looking for a solution online
* collaboration with peers
* outsourcing.

|  |
| --- |
|  |

## Propose an additional innovative solution using a prototype and user interface (UI) design

In the space below [sketch a wireframe](https://designshack.net/articles/graphics/what-is-a-wireframe/) of your additional innovative solution:

|  |
| --- |
|  |

# 4. Testing and evaluating

## Apply methodologies to test and evaluate code

Software engineers apply various methodologies and techniques to test and evaluate code to ensure its quality, reliability, and functionality.

Research each of these methodologies:

* functional testing
* acceptance testing
* live data
* simulated data
* beta testing
* volume testing.

Suggest an analogy to complete the table of common methodologies.

|  |  |
| --- | --- |
| Methodologies to test and evaluate code | Descriptive analogy |
| testing | [Insert descriptive analogy.] |
| functional testing | Is like checking if a car works as it should. Just as you would test if the car can start, accelerate, brake and turn, functional testing checks if each part of a software system works correctly. It's about making sure that the software does what it's supposed to do, like allowing users to log in, search for items and make purchases on an online store. |
| acceptance testing | Is similar to checking if a newly built house meets the homeowner's expectations. Just as the homeowner walks through the house to make sure everything is as they wanted, acceptance testing checks if the software behaves as the users expect it to. It is the final phase of testing before the software is released to the users. |
| live data testing | Is like testing a recipe with real ingredients instead of just imagining how it will taste. Just as using real ingredients helps you understand how the dish will turn out, live data testing helps testers understand how the software will perform in real-world scenarios using actual data. |
| simulated data testing | Is similar to testing a car by using a driving simulator instead of driving it on real roads. Just as a driving simulator helps you understand how a car performs in different conditions, simulated data testing helps testers understand how the software behaves with different types of data. |
| beta testing | Is like asking a few friends to try out a new recipe before serving it to a larger group. Just as you would get feedback on the taste and texture of the dish, beta testing gathers feedback from users on the software's performance, usability and overall experience. |
| volume testing | Is similar to testing how a car performs with a full tank of fuel and a full load of passengers and luggage. Just as you would want to ensure the car can handle the extra weight and fuel, volume testing checks if the software can handle a large amount of data without slowing down or crashing. |

Highlight the methodologies to test and evaluate code that are most relevant to your project.

## Use a language-dependent code optimisation technique.

Software engineers use various language-dependent code optimisation techniques to improve the performance, efficiency, and maintainability of their code. These techniques are specific to the programming language being used and take advantage of language features and characteristics.

The table below provides some common language-dependent code optimisation techniques.

This should be issued blank to students to complete after the following jigsaw research activity. The jigsaw activity should be adapted to languages that students are familiar with.

|  |  |
| --- | --- |
| Common language-dependent code optimisation techniques | Description |
| C/C++ optimisations | Use of inline functions to reduce function call overhead.  Compiler optimisations such as loop unrolling, constant folding, and dead code elimination.  Efficient memory management using techniques like stack allocation and memory pooling. |
| Java optimisations | Just-In-Time (JIT) compilation to convert Java bytecode into native machine code at runtime for better performance.  Use of the Java Virtual Machine (JVM) optimisations such as HotSpot to dynamically optimise code based on runtime profiling. |
| Python optimisations | Utilising built-in functions and data structures (for example, list comprehensions, dictionaries) for more efficient code.  Using libraries like NumPy for numerical computations, which are optimised for performance. |

****[**Jigsaw activity**](https://app.education.nsw.gov.au/digital-learning-selector/): code optimisation techniques.

Jigsaw is a cooperative learning strategy that:

* supports educators to differentiate learning
* enables each student to specialise in one aspect of a topic, with each student's part being essential to the completion of the task
* builds students' comprehension, cooperation, communication and problem-solving skills.

Assign students into ’home groups’.

There should be **3** students in each home group.

Students in each home group are numbered 1, 2,3.

* All students in home group 1 form an expert group that will research C/C++ optimisations
* All students in home group 2 form an expert group that will research Java optimisations
* All students in home group 3 form an expert group that will research Python optimisations
* Each group completes the research task below and returns to the home group to teach the other group members about the software development approach they have researched.

|  |  |
| --- | --- |
| Expert Group 1 C/C++ optimisations | Explanation and sample code |
| Use of inline functions to reduce function call overhead. | [Insert explanation and sample code] |
| Compiler optimisations such as loop unrolling, constant folding, and dead code elimination. | [Insert explanation and sample code] |
| Efficient memory management using techniques like stack allocation and memory pooling. | [Insert explanation and sample code] |

|  |  |
| --- | --- |
| Expert Group 2 Java optimisations | Explanation and sample code |
| Just-In-Time (JIT) compilation to convert Java bytecode into native machine code at runtime for better performance. | [Insert explanation and sample code] |
| Use of the Java Virtual Machine (JVM) optimizations such as HotSpot to dynamically optimise code based on runtime profiling. | [Insert explanation and sample code] |

|  |  |
| --- | --- |
| Expert Group 3 Python optimisations | Explanation and sample code |
| Utilising built-in functions and data structures (for example, list comprehensions and dictionaries) for more efficient code. | [Insert explanation and sample code] |
| Using libraries like NumPy for numerical computations, which are optimised for performance. | [Insert explanation and sample code] |

* Expert group members return to their ‘home groups’ and peer instruct the other members of the home group on their research findings.
* Each ‘home group’ presents to class.
* Students complete the code optimisation table above according to their presentations.

## Analyse and respond to feedback

Students analyse feedback given to them on the software solution they have just created. This feedback can be in the form of an interview, survey, focus group, observation of the Q&A forum and may be from the client, peer review, critical friend or the teacher.

## Evaluate the effectiveness of a software engineering solution

Analysis of a solution against quality success criteriaStudents are to take each quality success criteria from Section 2 and place it in the table below. For each quality criteria, **analyse** the components of the solution that met or did not meet each quality criteria. Give reasons why each success criteria were or were not met.

|  |  |  |
| --- | --- | --- |
| Quality criteria | Met? | Analysis |
|  |  |  |
|  |  |  |
|  |  |  |

### Developing a report to synthesise feedback

Students collate all feedback and synthesise this into a report that categorises common themes including:

* usability
* performance conclusions
* recommendations.

In the space provided below **provide a summary of the feedback** that includes overall positive, and negative sentiments towards their software solution in their response.

|  |
| --- |
|  |

### Developing a test plan

Developing a test plan is crucial for evaluating the effectiveness of a software engineering solution by ensuring that the software is tested thoroughly, consistently, and in accordance with its requirements.

The test plan is a living document that can be updated and refined throughout the software development lifecycle. Software engineers use feedback from testing to update the test plan and continuously improve the effectiveness of the software engineering solution.

A test plan should:

* identify the resources required for testing, such as testing tools and environments
* define the criteria for evaluating the results of the tests, such as pass/fail criteria, acceptance criteria, and performance. This provides a basis for determining whether the software meets its requirements.
* include detailed test cases that describe the steps to be taken, the expected results, and the criteria for determining success or failure ensuring that the tests are conducted consistently and accurately
* specify the schedule and sequence for executing the tests. By following the test plan, testers can ensure that all necessary tests are conducted and that the results are recorded and analysed.

### Comparing actual output with expected output

Once the tests are completed results are analysed. This involves comparing the actual results to the expected results, identifying any discrepancies, and determining the cause of any failures.

The test plan includes a reporting mechanism for documenting the results of the tests. This helps stakeholders understand the effectiveness of the software engineering solution and any areas that may require further improvement.

### Testing data used/generated based on path and boundary testing

Students **identify** variables and values which were used for either path and/or boundary testing.

Students **develop** these test data tables based on their algorithms versus their real code.

Students then **state** the reason for including these variables.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Variable | Maximum | Minimum | Default value | Expected output | Actual output | Reason for inclusion |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |

Boundary testing comparing actual output with expected output.

# References

This resource contains NSW Curriculum and syllabus content. The NSW Curriculum is developed by the NSW Education Standards Authority. This content is prepared by NESA for and on behalf of the Crown in right of the State of New South Wales. The material is protected by Crown copyright.

Please refer to the NESA Copyright Disclaimer for more information <https://educationstandards.nsw.edu.au/wps/portal/nesa/mini-footer/copyright>.

NESA holds the only official and up-to-date versions of the NSW Curriculum and syllabus documents. Please visit the NSW Education Standards Authority (NESA) website <https://educationstandards.nsw.edu.au> and the NSW Curriculum website <https://curriculum.nsw.edu.au>.

[Software Engineering 11–12 Syllabus](https://curriculum.nsw.edu.au/learning-areas/tas/software-engineering-11-12-2022/overview) © NSW Education Standards Authority (NESA) for and on behalf of the Crown in right of the State of New South Wales, 2022.

[Higher School Certificate Course Specifications – Software Engineering](https://curriculum.nsw.edu.au/learning-areas/tas/software-engineering-11-12-2022/overview#software-engineering-course-specifications-software_engineering_11_12_2022) © NSW Education Standards Authority (NESA) for and on behalf of the Crown in right of the State of New South Wales, 2023.

International Organization for Standardization (2022), [ISO/IEC 27001: 2022 Information security, cybersecurity and privacy protection: Information security management systems](https://www.iso.org/standard/27001), ISO website, accessed 10 April 2024.

Lucid Software Inc (2024) [Decision tree](https://www.lucidchart.com/pages), Lucidchart website, accessed 3 April 2024.

Lucid Software Inc (2024) [*Flowchart Symbols and Notation*](https://www.lucidchart.com/pages/flowchart-symbols-meaning-explained?a=0), Lucidchart website, accessed 3 April 2024.

Lucid Software Inc (2024) [How to Make a Decision Tree Diagram](https://www.lucidchart.com/pages/how-to-make-a-decision-tree-diagram), Lucidchart website, accessed 3 April 2024.

Miro (2024) [Data Flow Diagram](https://miro.com/diagramming/data-flow-diagram/), Miro website, accessed 3 April 2024.

Miro (2024) [What is a data flow diagram?](https://miro.com/diagramming/what-is-a-data-flow-diagram/), Miro website, accessed 3 April 2024.

[Privacy Act 1988](https://www.legislation.gov.au/Series/C2004A03712), Federal Register of Legislation website, accessed 10 April 2024.

[Privacy and Personal Information Protection Act 1998](https://legislation.nsw.gov.au/view/whole/html/inforce/current/act-1998-133#statusinformation), NSW legislation website, accessed 10 April 2024.

**© State of New South Wales (Department of Education), 2024**

The copyright material published in this resource is subject to the Copyright Act 1968 (Cth) and is owned by the NSW Department of Education or, where indicated, by a party other than the NSW Department of Education (third-party material).

Copyright material available in this resource and owned by the NSW Department of Education is licensed under a [Creative Commons Attribution 4.0 International (CC BY 4.0) license](https://creativecommons.org/licenses/by/4.0/).

[](https://creativecommons.org/licenses/by/4.0/)

This license allows you to share and adapt the material for any purpose, even commercially.

Attribution should be given to © State of New South Wales (Department of Education), 2024.

Material in this resource not available under a Creative Commons license:

* the NSW Department of Education logo, other logos and trademark-protected material
* material owned by a third party that has been reproduced with permission. You will need to obtain permission from the third party to reuse its material.

**Links to third-party material and websites**

Please note that the provided (reading/viewing material/list/links/texts) are a suggestion only and implies no endorsement, by the New South Wales Department of Education, of any author, publisher, or book title. School principals and teachers are best placed to assess the suitability of resources that would complement the curriculum and reflect the needs and interests of their students.

If you use the links provided in this document to access a third-party's website, you acknowledge that the terms of use, including licence terms set out on the third-party's website apply to the use which may be made of the materials on that third-party website or where permitted by the Copyright Act 1968 (Cth). The department accepts no responsibility for content on third-party websites.