NSW Education Standards Authority

Software Engineering 11–12 (2022)

curriculum.nsw.edu.au

Generated on Nov 2022

Table of contents

Software Engineering 11–12 (2022)

Implementation from 2024

The new Software Engineering 11–12 Syllabus (2022) is to be taught from 2024.

2023

- Plan and prepare to teach the new syllabus

2024, Term 1

- Start teaching new syllabus for Year 11
- Start implementing new Year 11 school-based assessment requirements
- Continue to teach the Software Design and Development Stage 6 Syllabus (2010) for Year 12

2024, Term 4

- Start teaching new syllabus for Year 12
- Start implementing new Year 12 school-based assessment requirements

2025

- First HSC examination for new syllabus

Course overview

Course structure and requirements

Course numbers:

- Software Engineering (Year 11, 2 units): TBA
- Software Engineering (Year 12, 2 units): TBA

Exclusions:

- Computing Technology Life Skills (Year 11, 2 units): TBA
- Computing Technology Life Skills (Year 12, 2 units): TBA

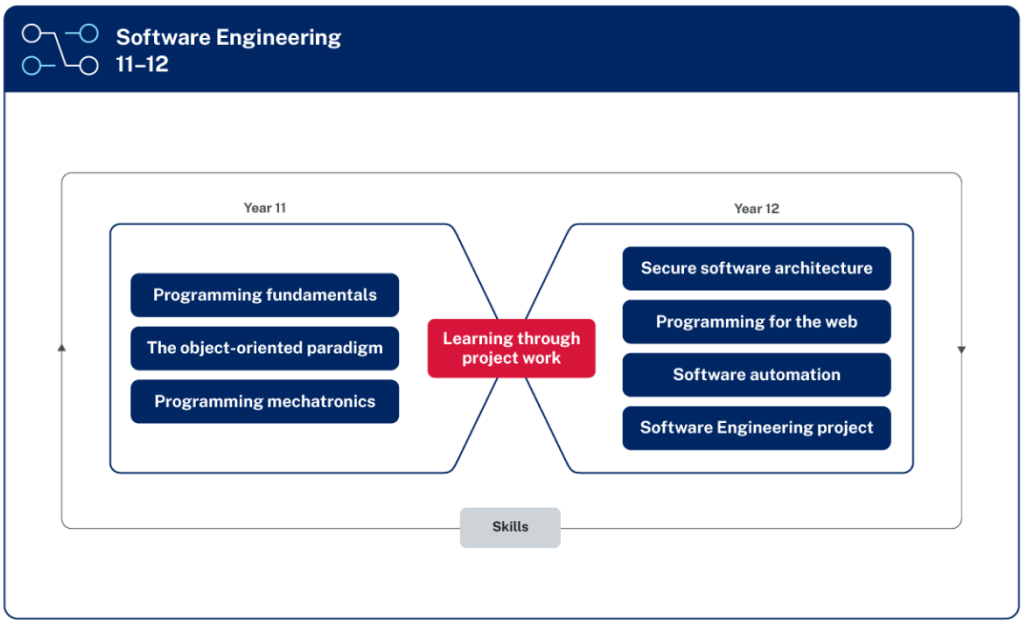

Figure 1: The organisation of content for Software Engineering 11–12 Syllabus

Image long description: This is a diagram outlining the organisation content for the Software Engineering 11–12 Syllabus. Content is listed in boxes, Programming fundamentals, The object-oriented paradigm, and Programming mechatronics fall under Year 11 on the left. Secure software architecture, Programming for the web, Software automation and Software Engineering project fall under Year 12 on the right. In the middle is Learning through project work, with lines coming from this box and encircling the content. Surrounding all content of the diagram is a line which is joined at the bottom by a box labelled, Skills. This demonstrates that Skills is being both developed and used in all focus areas.

Year 11 course structure and requirements

The Year 11 course provides students with opportunities to develop and apply an understanding of the fundamental elements involved in creating software.

Year 11 course (120 hours)

Year 11	Indicative hours
Programming Fundamentals	40
The Object-Oriented Paradigm	40
Programming Mechatronics	40

Year 12 course structure and requirements

The Year 12 course provides students with opportunities to extend their knowledge, understanding and skills in the development of software. A major software engineering project provides students with the opportunity to further develop project management skills.

Year 12 course (120 hours)

Year 12	Indicative hours
Secure Software Architecture	30
Programming for the Web	30
Software Automation	30
Software Engineering Project	30

Software Engineering course specifications

The Software Engineering Course Specifications are an integral part of the course content for Year 11 and Year 12 and indicate the depth of study required for some concepts in the *Software Engineering 11–12 Syllabus*. The *Software Engineering 11–12 Syllabus* must be applied in conjunction with the Software Engineering Course Specifications (available December 2022).

Rationale

The study of *Software Engineering 11–12* enables students to develop an understanding of software engineering as a facet of computer science. Students have the opportunity to develop knowledge and understanding of software engineering, hardware and software integration, and the development, implementation and evaluation of computer programs. They focus on a systematic approach to problem-solving when designing and developing creative software solutions.

Software Engineering promotes a deeper understanding of fundamental concepts, programming languages and innovative technologies, leading to greater flexibility when developing software solutions. Students perform project work and apply their knowledge and skills in: programming fundamentals, the object-oriented paradigm, programming mechatronics, secure software architecture, programming for the web and software automation, and use the acquired knowledge and skills to develop a software engineering project. Project work enables students to collaborate on problems and develop team and communication skills that are highly valued in the industry.

Software Engineering encourages students to explore the impact of innovations in computing technology on society and the environment. They engage with technologies that improve access to, and participation in, a range of industries.

The *Software Engineering 11–12 Syllabus* provides students with the opportunity to develop their computing skills across 4 domains: technical skills, social awareness, project management and thinking skills. Students are encouraged to transfer knowledge to new situations and projects, building on technical skills and past learning. They enhance their understanding of project management through collaboration, communicating ideas, engaging in processes and designing solutions.

Aim

The aim of Software Engineering is to develop in each student:

- a capacity to think creatively to develop and program software solutions
- an ability to apply knowledge, understanding and thinking skills to develop and communicate solutions to real-world problems.

Table of outcomes

Year 11	Year 12
SE-11-01 describes methods used to plan, develop and engineer software solutions	**SE-12-01** justifies methods used to plan, develop and engineer software solutions
SE-11-02 explains how structural elements are used to develop programming code	**SE-12-02** applies structural elements to develop programming code
SE-11-03 describes how current hardware, software and emerging technologies influence the development of software engineering solutions	**SE-12-03** analyses how current hardware, software and emerging technologies influence the development of software engineering solutions
SE-11-04 applies safe and secure practices to collect, use and store data	**SE-12-04** evaluates practices to safely and securely collect, use and store data
SE-11-05 describes the social, ethical and legal implications of software engineering on the individual, society and the environment	**SE-12-05** explains the social, ethical and legal implications of software engineering on the individual, society and the environment
SE-11-06 applies tools and resources to design, develop, manage and evaluate software	**SE-12-06** justifies the selection and use of tools and resources to design, develop, manage and evaluate software
SE-11-07 implements safe and secure programming solutions	**SE-12-07** designs, develops and implements safe and secure programming solutions
SE-11-08 applies language structures to refine code	**SE-12-08** tests and evaluates language structures to refine code
SE-11-09 manages and documents the development of a software project	**SE-12-09** applies methods to manage and document the development of a software project

Outcomes and content for Year 11

Programming fundamentals

Outcomes

A student:

- describes methods used to plan, develop and engineer software solutions **SE-11-01**
- explains how structural elements are used to develop programming code **SE-11-02**
- describes how current hardware, software and emerging technologies influence the development of software engineering solutions **SE-11-03**
- applies safe and secure practices to collect, use and store data **SE-11-04**
- applies tools and resources to design, develop, manage and evaluate software **SE-11-06**
- implements safe and secure programming solutions **SE-11-07**

Content

Software development

- Explore fundamental software development steps used by programmers when designing software

  Including:
  - requirements definition
  - determining specifications
  - design
  - development
  - integration
  - testing and debugging
  - installation
  - maintenance
- Research and evaluate the prevalence and use of online code collaboration tools

Designing algorithms

- Apply computational thinking and algorithmic design by defining the key features of standard algorithms, including sequence, selection, iteration and identifying data that should be stored
- Apply divide and conquer and backtracking as algorithmic design strategies
- Develop structured algorithms using pseudocode and flowcharts, including the use of subprograms
- Use modelling tools including structure charts, abstraction and refinement diagrams to support top-down and bottom-up design
- Analyse the logic and structure of written algorithms

  Including:
  - determining inputs and outputs
  - determining the purpose of the algorithm
  - desk checking and peer checking
  - determining connections of written algorithms to other subroutines or functions
- Identify procedures and functions in an algorithm
- Experiment with object-oriented programming, imperative, logic and functional programming paradigms

Data for software engineering

- Investigate the use of number systems for computing purposes, including binary, decimal and hexadecimal
- Represent integers using two's complement
- Investigate standard data types

  Including:
  - char (character) and string
  - Boolean
  - real
  - single precision floating point
  - integer
  - date and time
- Create data dictionaries as a tool to describe data and data types, structure data, and record relationships
- Use data structures of arrays, records, trees and sequential files

Developing solutions with code

- Apply skills in computational thinking and programming to develop a software solution

  Including:
  - converting an algorithm into code
  - using control structures
  - using data structures
  - using standard modules
  - creating relevant subprograms that incorporate parameter passing
- Implement data structures that support data storage

  Including:
  - single and multidimensional arrays
  - lists
  - trees
  - stacks
  - hash tables
- Compare the execution of the Waterfall and Agile project management models as applied to software development
- Test and evaluate solutions, considering key aspects including functionality, performance, readability of code, quality of documentation
- Use debugging tools

  Including:
  - breakpoints
  - single line stepping
  - watches
  - interfaces between functions
  - debugging output statements
  - debugging software available in an integrated development environment (IDE)

- Determine sets of suitable test data

  Including:
  - boundary values
  - path coverage
  - faulty and abnormal data
- Determine typical errors experienced when developing code, including syntax, logic and runtime, and explain their likely causes

The object-oriented paradigm

Outcomes

A student:

- describes methods used to plan, develop and engineer software solutions **SE-11-01**
- explains how structural elements are used to develop programming code **SE-11-02**
- describes how current hardware, software and emerging technologies influence the development of software engineering solutions **SE-11-03**
- applies safe and secure practices to collect, use and store data **SE-11-04**
- applies tools and resources to design, develop, manage and evaluate software **SE-11-06**
- implements safe and secure programming solutions **SE-11-07**
- applies language structures to refine code **SE-11-08**
- manages and documents the development of a software project **SE-11-09**

Content

Understanding OOP

- Apply the key features of an object-oriented programming (OOP) language

  Including:
  - objects
  - classes
  - encapsulation
  - abstraction
  - inheritance
  - generalisation
  - polymorphism
- Compare procedural programming with OOP
- Use data flow diagrams, structure charts and class diagrams to represent a system
- Describe the process of design used to develop code in an OOP language

  Including:
  - task definition
  - top-down and bottom-up
  - facade pattern
  - agility
- Assess the effectiveness of programming code developed to implement an algorithm
- Investigate how OOP languages handle message-passing between objects
- Explain code optimisation in software engineering
- Outline the features of OOP that support collaborative code development

  Including:
  - consistency
  - code commenting
  - version control
  - feedback

Programming in OOP

- Design and implement computer programs involving branching, iteration and functions in an OOP language for an identified need or opportunity

- Implement and modify OOP programming code

  Including:
  - clear and uncluttered mainline
  - one logical task per subroutine
  - use of stubs
  - use of control structures and data structures
  - ease of maintenance
  - version control
  - regular backup

- Apply methodologies to test and evaluate code

  Including:
  - unit, subsystem and system testing
  - black, white and grey box testing
  - quality assurance

Programming mechatronics

Outcomes

A student:

- describes methods used to plan, develop and engineer software solutions **SE-11-01**
- explains how structural elements are used to develop programming code **SE-11-02**
- describes how current hardware, software and emerging technologies influence the development of software engineering solutions **SE-11-03**
- applies safe and secure practices to collect, use and store data **SE-11-04**
- describes the social, ethical and legal implications of software engineering on the individual, society and the environment **SE-11-05**
- applies tools and resources to design, develop, manage and evaluate software **SE-11-06**
- implements safe and secure programming solutions **SE-11-07**
- applies language structures to refine code **SE-11-08**
- manages and documents the development of a software project **SE-11-09**

Content

Understanding mechatronic hardware and software

- Outline applications of mechatronic systems in a variety of specialised fields
- Identify the hardware requirements to run a program and the effect on code development

  Including:
  - assessing the relationship of microcontrollers and the central processing unit (CPU)
  - the influence of instruction set and opcodes
  - the use of address and data registers

- Identify and describe a range of sensors, actuators and end effectors/manipulators within existing mechatronic systems

  Including:
  - motion sensors
  - light level sensors
  - hydraulic actuators
  - robotic grippers

- Use different types of data and understand how it is obtained and processed in a mechatronic system, including diagnostic data and data used for optimisation
- Experiment with software to control interactions and dependencies within mechatronic systems

  Including:
  - motion constraints
  - degrees of freedom
  - combination of subsystems
  - combination of sensors, actuators and end effectors to create viable subsystems

- Determine power, battery and material requirements for components of a mechatronic system
- Develop wiring diagrams for a mechatronic system, considering data and power supply requirements
- Determine specialist requirements that influence the design and functions of mechatronic systems designed for people with disability

Designing control algorithms

- Develop, modify and apply algorithms to control a mechatronic system
- Explore the algorithmic patterns, code and applications for open and closed control systems
- Outline the features of an algorithm and program code used for autonomous control

Programming and building

- Design, develop and produce a mechatronic system for a real-world problem

  Including:
  - software control
  - mechanical engineering
  - electronics and mathematics
- Implement algorithms and design programming code to drive mechatronic devices
- Develop simulations and prototypes of a potential mechatronic system to test programming code
- Design, develop and implement programming code for a closed loop control system
- Apply programming code to integrate sensors, actuators and end effectors/manipulators
- Implement specific control algorithms that enhance the performance of a mechatronic system
- Design, develop and implement a user interface (UI) to control a mechatronic system
- Create and use unit tests to determine the effectiveness and repeatability of each component's control algorithm

Outcomes and content for Year 12

Secure software architecture

Outcomes

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**

Content

Designing software

- Describe the benefits of developing secure software

  Including:
  - data protection
  - minimising cyber attacks and vulnerabilities

- Interpret and apply fundamental software development steps to develop secure code

  Including:
  - requirements definition
  - determining specifications
  - design
  - development
  - integration
  - testing and debugging
  - installation
  - maintenance

- Describe how the capabilities and experience of end users influence the secure design features of software

Developing secure code

- Explore fundamental software design security concepts when developing programming code

  Including:
  - confidentiality
  - integrity
  - availability
  - authentication
  - authorisation
  - accountability
- Apply security features incorporated into software including data protection, security, privacy and regulatory compliance
- Use and explain the contribution of cryptography and sandboxing to the ‘security by design’ approach in the development of software solutions
- Use and explain the ‘privacy by design’ approach in the development of software solutions

  Including:
  - proactive not reactive approach
  - embed privacy into design
  - respect for user privacy
- Test and evaluate the security and resilience of software by determining vulnerabilities, hardening systems, handling breaches, maintaining business continuity and conducting disaster recovery
- Apply and evaluate strategies used by software developers to manage the security of programming code

  Including:
  - code review
  - static application security testing (SAST)
  - dynamic application security testing (DAST)
  - vulnerability assessment
  - penetration testing
- Design, develop and implement code using defensive data input handling practices, including input validation, sanitisation and error handling
- Design, develop and implement a safe application programming interface (API) to minimise software vulnerabilities
- Design, develop and implement code considering efficient execution for the user

  Including:
  - memory management
  - session management
  - exception management
- Design, develop and implement secure code to minimise vulnerabilities in user action controls

  Including:
  - broken authentication and session management
  - cross-site scripting (XSS) and cross-site request forgery (CSRF)
  - invalid forwarding and redirecting
  - race conditions
- Design, develop and implement secure code to protect user file and hardware vulnerabilities from file attacks and side channel attacks

Impact of safe and secure software development

- Apply and describe the benefits of collaboration to develop safe and secure software

  Including:
  - considering various points of view
  - delegating tasks based on expertise
  - quality of the solution

- Investigate and explain the benefits to an enterprise of the implementation of safe and secure development practices

  Including:
  - improved products or services
  - influence on future software development
  - improved work practices
  - productivity
  - business interactivity

- Evaluate the social, ethical and legal issues and ramifications that affect people and enterprises resulting from the development and implementation of safe and secure software

  Including:
  - employment
  - data security
  - privacy
  - copyright
  - intellectual property
  - digital disruption

Programming for the web

Outcomes

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**

Content

Data transmission using the web

- Explore the applications of web programming

  Including:
  - interactive website/webpages
  - e-commerce
  - progressive web apps (PWAs)

- Investigate and practise how data is transferred on the internet

  Including:
  - data packets
  - internet protocol (IP) addresses, including IPv4
  - domain name systems (DNS)

- Investigate and describe the function of web protocols and their ports

  Including:
  - HTTP, HTTPS
  - TCP/IP
  - DNS
  - FTP, SFTP
  - SSL, TLS
  - SMTP, POP 3, IMAP

- Explain the processes for securing the web

  Including:
  - Secure Sockets Layer (SSL) certificates
  - encryption algorithms
  - encryption keys
  - plain text and cipher text
  - authentication and authorisation
  - hash values
  - digital signatures

- Investigate the effect of big data on web architecture

  Including:
  - data mining
  - metadata
  - streaming service management

Designing web applications

- Investigate and explain the role of the World Wide Web Consortium (W3C) in the development of applications for the web

  Including:
  - Web Accessibility Initiative (WAI)
  - internationalisation
  - web security
  - privacy
  - machine-readable data

- Model elements that form a web development system

  Including:
  - client-side (front-end) web programming
  - server-side (back-end) web programming
  - interfacing with databases that are based on Structured Query Language (SQL) or non-SQL

- Explore and explain the influence of a web browser on web development, including the use of developer (dev) tools
- Investigate cascading style sheets (CSS) and its impact on the design of a web application

  Including:
  - consistency of appearance
  - flexibility with browsers or display devices
  - CSS maintenance tools

- Investigate the reasons for version control and apply it when developing web application
- Explore the types and significance of code libraries for front-end web development

  Including:
  - frameworks that control complex web applications
  - template engines
  - predesigned CSS classes

- Explain the use and development of open-source software in relation to web development
- Investigate methods to support and manage the load times of web pages/applications

- Research, experiment with and evaluate the prevalence and use of web content management systems (CMS)
- Assess the contribution of back-end web development to the success of a web application
- Observe and describe the back-end process used to manage a web request

  Including:
  - role of webserver software
  - web framework
  - objects
  - libraries
  - databases
- Develop a web application using an appropriate scripting language with shell scripts to make files and directories, and searching for text in a text file
- Apply a web-based database and construct script that executes SQL

  Including:
  - selecting fields
  - incorporating ‘group by’
  - common SQL queries
  - constraints using WHERE keyword
  - table joins
- Compare Object-Relational Mapping (ORM) to SQL
- Describe how collaborative work practices between front-end and back-end developers improve the development of a web solution
- Design, develop and implement a progressive web app (PWA)

  Including:
  - the application of design and user interface (UI) and user experience (UX) principles of font, colour, audio, video and navigation
  - a UI that considers accessibility and inclusivity

Software automation

Outcomes

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**

Content

Algorithms in machine learning

- Investigate how machine learning (ML) supports automation through the use of DevOps, robotic process automation (RPA) and business process automation (BPA)
- Distinguish between artificial intelligence (AI) and ML
- Explore models of training ML

  Including:
  - supervised learning
  - unsupervised learning
  - semi-supervised learning
  - reinforcement learning

- Investigate common applications of key ML algorithms

  Including:
  - data analysis and forecasting
  - virtual personal assistants
  - image recognition

- Research models used by software engineers to design and analyse ML

  Including:
  - decision trees
  - neural networks

- Describe types of algorithms associated with ML

  Including:
  - linear regression
  - logistic regression
  - K-nearest neighbour

Programming for automation

- Design, develop and apply ML regression models using an OOP to predict numeric values

  Including:
  - linear regression
  - polynomial regression
  - logistic regression
- Apply neural network models using an OOP to make predictions

Significance and impact of ML and AI

- Assess the impact of automation on the individual, society and the environment

  Including:
  - safety of workers
  - people with disability
  - the nature and skills required for employment
  - production efficiency, waste and the environment
  - the economy and distribution of wealth
- Explore by implementation how patterns in human behaviour influence ML and AI software development

  Including:
  - psychological responses
  - patterns related to acute stress response
  - cultural protocols
  - belief systems
- Investigate the effect of human and dataset source bias in the development of ML and AI solutions

Software engineering project

Outcomes

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**

Content

Identifying and defining

- Define and analyse the requirements of a problem

  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

- Explore tools used to develop ideas and generate solutions

  Including:
  - brainstorming, mind-mapping and storyboards
  - data dictionaries, including selecting appropriate data types
  - algorithm design
  - code generation
  - testing and debugging
  - installation
  - maintenance

- Investigate types of software implementation methods

  Including:
  - direct
  - phased
  - parallel
  - pilot

Research and planning

- Research and use the Waterfall software development approach

  Including:
  - logical progression of steps used throughout the life cycle
  - stages of ‘falling water’
  - advantages and disadvantages
  - scale and types of developments

- Research and use the Agile software development approach

  Including:
  - rate of developing a final solution
  - method tailoring
  - iteration workflow
  - scale and types of developments

- Research the WAgile software development approach

  Including:
  - understanding it is a hybrid model
  - analysis of the ‘when’ and ‘how’ intervention is applied during the development life cycle
  - scale and types of developments

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

  Including:
  - scheduling and tracking using a software tool, including Gantt charts
  - using collaboration tools

- Explore social and ethical issues associated with project work, including working individually, collaboratively and responding to stakeholders
- 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

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

  Including:
  - technology used
  - error handling
  - interfacing with front end
  - security engineering

Producing and implementing

- Design, construct and implement a solution to a software problem using appropriate development approach(es)
- Present a software engineering solution using presentation software
- Develop, construct and document algorithms
- Allocate resources to support the development of a software engineering solution
- Demonstrate the use of programmed data backup
- Implement version control when developing a software engineering solution
- Explore strategies to respond to difficulties when developing a software engineering solution

  Including:
  - looking for a solution online
  - collaboration with peers
  - outsourcing
- 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
- Analyse and respond to feedback
- Evaluate the effectiveness of a software engineering solution

  Including:
  - developing a report to synthesise feedback
  - developing a test plan
  - testing data used/generated based on path and boundary testing
  - comparing actual output with expected output