9 Content: Software Design and Development Stage 6 HSC Course

9.1 Development and Impact of Software Solutions

9.1.1 Social and ethical issues

Students undertaking the HSC course should be aware of the broader social and ethical issues associated with the development and use of software.

This topic builds on the concepts covered in the Preliminary course and looks specifically at the rights and responsibilities of developers from a number of perspectives. Both past and current problems arising from the use of software are investigated to illustrate the effects on society of these and similar problems.

Outcomes

A student:

H2.2 explains the relationship between emerging technologies and software development

H3.1 identifies and evaluates legal, social and ethical issues in a number of contexts.

Students learn about:	Students learn to:
The impact of software • inappropriate data structures, for example the year 2000 problem • computer malware such as viruses • reliance on software • social networking • cyber safety • huge amounts of information (which may be unsupported, unverifiable, misleading or incorrect) available through the internet	• recognise the effects of software solutions on society • identify the impact of inappropriately developed software on users • identify the effect of the inappropriate use of software on society and individuals
Rights and responsibilities of software developers • acknowledging the intellectual property of others • recognition by others of the developer's intellectual property • producing quality software solutions • appropriately responding to user-identified problems • adhering to code of conduct • neither generating nor transmitting malware • addressing ergonomic issues in software design • ensuring software addresses inclusivity issues • ensuring individuals' privacy is not compromised	• apply a relevant code of conduct to their own software development
Software piracy and copyright • concepts associated with piracy and copyright, including: – intellectual property – plagiarism – copyright laws – licensing issues – licence conditions – shareware – public domain – open source – ownership versus licensing – collaboratively developed software – reverse engineering – decompilation	• interpret licence agreements and develop personal practices that reflect current laws • identify the relationship between copyright laws and software license agreements • acknowledge all sources in recognition of the intellectual contribution of authors
• current and emerging technologies used to combat software piracy (see Course Specifications document)	• identify a range of techniques designed to combat software piracy

Students learn about:	**Students learn to:**

Use of networks
• by the developer when developing software
  – access to resources
  – ease of communication
  – productivity
 	
• by the user when using network based software
  – response times
  – interface design
  – privacy and security issues
	• evaluate the usefulness of networks in the development environment

The software market • maintaining market position • the effect of dominant developers of software • the impact of new developers of software and new products	• identify the impact of dominant developers of software on software development
Legal implications • national and international legal action resulting from software development (see Course Specifications document)	• discuss the reasons for, and consequences of, significant legal actions pertaining to the development of software

9.1.2 Application of software development approaches

Students should be aware of the appropriateness of each of the different software development approaches for a given situation. In this topic, students complete a case study of a software solution. In so doing, students will engage in a real-world investigation of a significant software solution.

Outcomes

A student:

H1.2	differentiates between various methods used to construct software solutions
H2.2	explains the interrelationship between emerging technologies and software development
H3.1	identifies and evaluates legal, social and ethical issues in a number of contexts
H4.2	applies appropriate development methods to solve software problems
H5.1	applies project management techniques to maximise the productivity of the software development
H5.2	creates and justifies the need for the various types of documentation required for a software solution
H5.3	selects and applies appropriate software to facilitate the design and development of software solutions
H6.1	assesses the skills required in the software development cycle
H6.2	communicates the processes involved in a software solution to an inexperienced user.

Students learn about:	Students learn to:
Software development approaches • approaches used in commercial systems, including: – Structured approach – Agile approach – Prototyping – RAD – End user approach – combinations of any of the above • use of Computer Aided Software Engineering (CASE) tools and their application in large systems development, including: – software version control – test data generation – production of documentation – production of code • methods of installation of new or updated systems – direct cut over – parallel – phased – pilot • employment trends in software development, for example: – outsourcing – contract programmers • trends in software development – changing nature of the environment in which developers work while creating software solutions – changing nature of applications (see Course Descriptions documents)	• compare and determine the most appropriate software development approach for a given scenario • communicate understanding of a commercial system studied using a case study approach by: – identifying the approaches used – discussing the appropriateness of the approaches used – describing how the various personnel contribute to the overall development – critically evaluating how social and ethical issues were addressed – evaluating how effectively the new system met the needs of the user • make informed comment on current trends in software development

9.2 Software Development Cycle

The formal methods that comprise the structured approach to software development empower students to undertake complex projects, knowing that the developed system will be robust and easily maintained.

The stages described in this topic should not be studied in isolation or in a sequential fashion. Students should be exposed to the content in a cyclic fashion and should recognise each stage during the development of their project(s). It is important that students are able to apply each of the stages in their project(s).

Areas for investigation in their project(s) could include writing scripts or code for modelling and simulation, games, scripted hypermedia products and applications.

9.2.1 Defining and understanding the problem

In order for students to be able to develop software to meet an identified need, they first need to be able to understand the specifications of a problem so that they can eventually translate these specifications into code.

As well as having good technical skills, it is necessary for students to have good communication skills so that the users' requirements can be fully understood and implemented throughout the development process. The modelling tools used should conform to those specified in the Software and Course Specifications document and should provide documentation that can be interpreted by developers and maintainers. Students should develop and refine skills as an integrated part of developing their software solutions. It is important at this initial stage of the process that all relevant social and ethical issues are considered as an integral part of the design and development of the solution.

Outcomes

A student:

H1.2	differentiates between various methods used to construct software solutions
H3.1	identifies and evaluates legal, social and ethical issues in a number of contexts
H3.2	constructs software solutions that address legal, social and ethical issues
H4.1	identifies needs to which software solutions are appropriate
H4.2	applies appropriate development methods to solve software problems
H4.3	applies a modular approach to implement well structured software solutions and evaluates their effectiveness
H5.1	applies project management techniques to maximise the productivity of the software development
H5.2	creates and justifies the need for the various types of documentation required for a software solution
H5.3	selects and applies appropriate software to facilitate the design and development of software solutions
H6.1	assesses the skills required in the software development cycle
H6.2	communicates the processes involved in a software solution to an inexperienced user
H6.3	uses and describes a collaborative approach during the software development cycle
H6.4	develops and evaluates effective user interfaces, in consultation with appropriate people.

Students learn about:	**Students learn to:**
Defining the problem • identifying the problem – needs of the client - functionality requirements - compatibility issues - performance issues – boundaries of the problem	
Issues relevant to a proposed solution • determining if an existing solution can be used – social and ethical considerations – consideration of existing software products – customisation of existing software solutions – cost effectiveness – licensing considerations	• evaluate the extent to which a proposed system will meet user needs • evaluate the effectiveness of using existing software
• selecting an appropriate development approach if there is no appropriate existing solution	• identify the parts of the proposed system that require software to be designed and developed • identify a relevant approach for a given problem
Design specifications • specifications of the proposed system • developer's perspective in consideration of: – data types – data structures – algorithms	
• user's perspective – interface design – social and ethical issues – relevance to the user's environment and computer configuration	• develop and interpret design specifications from a user's perspective • recognise the difference between the user's and developer's perspectives and the communication issues that may arise
System documentation • representing a system using systems modeling tools, including: – IPO diagrams – context diagrams – data flow diagrams (DFDs) – storyboards – structure charts – system flowcharts – data dictionaries • algorithms used to document the logic in modules and subroutines • test data and expected output	• differentiate between forms of systems documentation and the purposes for which each is used • describe a system by interpreting its diagrammatic representation • create a diagrammatic representation for a system using appropriate modeling tools
Communication issues between client and developer • the need to consult with the client • the need to incorporate the client's perspective • the need for the developer to enable and consider feedback • the need to involve and empower the client during the development process	• effectively communicate with users regarding a proposed software solution

Students learn about:	Students learn to:
Quality assurance • the need to explicitly define the criteria on which the quality of the product will be judged • putting in place management processes to ensure that quality criteria will be met • an ongoing process throughout development to ensure the quality criteria will be met	• identify a range of criteria on which the quality of the product will be judged • identify relevant processes for a given criterion that will result in a quality product

9.2.2 Planning and designing software solutions

To solve complex problems, students need to develop a strategy. They need to be able to identify inputs and outputs, to select, describe and use relevant data structures, to explain the procedures required for the solution and explain how each of these will interact. Well-structured algorithms should be developed. Desk checking of algorithms and documentation of the proposed solution are also important.

The development of structured algorithms to document the logical solution of problems is a fundamental principle of this course. These must be developed independently of any coding language. Students should appreciate that the real skill is in the development of the algorithm, not the implementation of the logic in a particular language. Not every algorithm developed in this section of the course need be implemented.

Problems must be chosen with an appropriate level of difficulty that reflects the ability level of students. The level of difficulty should be greater than in the Preliminary course. Relevant problems could include the development of games such as hangman, quizzes, mastermind, draughts and search-a-word. These problems should include use of data structures such as arrays of records and multidimensional arrays. Students should experience the storing, retrieving and updating of data in files.

Outcomes

A student:

H1.1	explains the interrelationship between hardware and software
H1.3	describes how the major components of a computer system store and manipulate data
H3.1	identifies and evaluates legal, social and ethical issues in a number of contexts
H3.2	constructs software solutions that address legal, social and ethical issues
H4.1	identifies needs to which software solutions are appropriate
H4.2	applies appropriate development methods to solve software problems
H4.3	applies a modular approach to implement well structured software solutions and evaluates their effectiveness
H5.1	applies project management techniques to maximise the productivity of the software development
H5.2	creates and justifies the need for the various types of documentation required for a software solution
H5.3	selects and applies appropriate software to facilitate the design and development of software solutions
H6.2	communicates the processes involved in a software solution to an inexperienced user
H6.3	uses and describes a collaborative approach during the software development cycle
H6.4	develops and evaluates effective user interfaces, in consultation with appropriate people.

Students learn about:	**Students learn to:**
Standard algorithms • standard logic used in software solutions, namely: – finding maximum and minimum values in arrays – processing strings (extracting, inserting, deleting) – generating a set of unique random numbers – processing of sequential files, including: - sentinel value - priming read - open for input, output or append - close - appending records – processing of relative files, including: - open for relative access - defining a key field for a relative file - retrieving, writing and updating a record in a relative file – linear search – binary search – bubble sort – insertion sort – selection sort (see Course Specifications document)	• recognise the logic in a standard approach, such as a sort or search • apply standard approaches as part of the solution to complex problems • read, interpret and modify algorithms developed by others • document the logic required to solve problems, including: – nesting of control structures – record structure – the use of files (sequential and relative) – random number generators – arrays of records – multidimensional arrays • develop a suitable set of test data • desk check algorithms and source code that include complex logic • select an appropriate data structure to solve a given problem
Custom-designed logic used in software solutions • requirements to generate these include: – identification of inputs, processes and outputs – representation as an algorithm – testing of the logic in the algorithm – identification and definition of required data structures – use of data structures, including multidimensional arrays, arrays of records, files (sequential and relative) (see Course Specifications document) • customised off-the-shelf packages – identifying an appropriate package – identifying the changes that need to be made – identifying how the changes are to be made	
Standard modules (library routines) used in software solutions • reasons for the development and use of standard modules • requirements for generating a module or subroutine for re-use, including: – identification of appropriate modules or subroutine – appropriate testing using drivers – thorough documentation of the routine: - author - date - purpose	• develop and appropriately document a module for use by others • correctly incorporate a standard module into a more complex solution, passing parameters effectively

Students learn about:	**Students learn to:**
- order and nature of parameters to be passed • issues associated with reusable modules or subroutines, including: – identifying appropriate modules or subroutines – considering local and global variables – appropriately using parameters (arguments)	
Documentation of the overall software solution • tools for representing a complex software solution, including: – algorithms – refined system modeling tools, including: - IPO diagrams - context diagrams - data flow diagrams (DFDs) - storyboards - structure charts - system flowcharts - data dictionaries	• represent a software solution in diagrammatic form • interpret and modify existing system modeling diagrams • select and use appropriate software to assist in the documentation of a software solution • recognise the relevance of CASE tools in the planning and design of a software solution
Interface design in software solutions • the design of individual screens in consultation with the client, including: – consideration of the intended audience – identification of screen size – identification of data fields and screen elements required and their appropriate on-screen placement – online help – consistency in approach – recognition of relevant social and ethical issues – current common practice in interface design (see Course Specifications document)	• design and evaluate effective interfaces for software solutions • use a RAD environment to produce user interfaces
Factors to be considered when selecting the programming language to be used • sequential or event-driven software – driven by the programmer or user • features required, and features available in the language • commands within the language to interface with the required hardware • ability to run under different operating systems	• recognise that the choice of programming language to be used depends on the problem to be solved
Factors to be considered when selecting the technology to be used • performance requirements • benchmarking	• interpret a benchmark report to select the most suitable technology for a specified task • produce a benchmark report for a simple iterative process running under two different environments or conditions

9.2.3 Implementation of software solution

In the implementation phase of the software development cycle, previously developed algorithms are converted to a form that can be processed by a computer. Students will need to learn the syntax of the language, macro or script being used to successfully implement their solutions. Knowledge of a metalanguage such as EBNF or railroad diagram(s) is essential in understanding both the syntax of a language and how a translator can detect syntax errors in source code. The need for a translation process should be recognized. In the case of code, students should be aware of the relevance of the different translation methods available. Students will need to recognise the approach being used (that is, sequential or event-driven) and will need to make appropriate decisions about the design of interfaces and the documentation produced. Relevant social and ethical issues should be considered during this implementation process.

Outcomes

A student:

H1.1	explains the interrelationship between hardware and software
H1.2	differentiates between various methods used to construct software solutions
H1.3	describes how the major components of a computer system store and manipulate data
H2.1	explains the implications of the development of different languages
H2.2	explains the interrelationship between emerging technologies and software development
H3.1	identifies and evaluates legal, social and ethical issues in a number of contexts
H3.2	constructs software solutions that address legal, social and ethical issues
H4.2	applies appropriate development methods to solve software problems
H4.3	applies a modular approach to implement well structured software solutions and evaluates their effectiveness
H5.1	applies project management techniques to maximise the productivity of the software development
H5.2	creates and justifies the need for the various types of documentation required for a software solution
H5.3	selects and applies appropriate software to facilitate the design and development of software solutions
H6.2	communicates the processes involved in a software solution to an inexperienced user
H6.3	uses and describes a collaborative approach during the software development cycle
H6.4	develops and evaluates effective user interfaces, in consultation with appropriate people.

Students learn about:	Students learn to:
Implementation of the design using an appropriate language • the different programming languages and the appropriateness of their use in solving different types of problems • construction of syntactically correct code that implements the logic described in the algorithm	• identify an appropriate language to solve a particular problem • recognise the appropriateness of either a sequential or event-driven approach to solve a particular problem • develop syntactically correct code to solve a problem in a given language
Language syntax required for software solutions • use of EBNF and railroad diagrams to describe the syntax of statements in the selected language	• interpret metalanguage definitions for commands in a selected language • produce syntactically correct statements using the metalanguage definitions • produce a generic metalanguage definition for a set of syntactically correct statements that use the same command • implement a solution from a complex algorithm using syntactically correct statements
The need for translational to machine code from source code • translation methods in software solutions including: – compilation – interpretation • advantages and disadvantages of each method • steps in the translation process – lexical analysis including token generation – syntactical analysis including parsing – code generation	• explain the use of tokens and the role of the parsing process during the translation of source code to machine code • recognise that machine code is the only code able to be executed by a computer • identify the most appropriate translation method for a given situation • use the features of both a compiler and an interpreter in the implementation of a software solution

Students learn about:	Students learn to:
The role of machine code in the execution of a program	
• machine code and CPU operation – instruction format – use of registers and accumulators – the fetch–execute cycle – use of a program counter and instruction register	• recognise, interpret and write machine code instructions for a problem fragment
• execution of called routines • linking, including use of DLLs	
Techniques used in developing well-written code	
• the use of good programming practice, including: – a clear and uncluttered mainline – one logical task per subroutine – use of stubs – appropriate use of control structures and data structures	• employ good programming practice when developing code • justify the use of a clear modular structure with separate routines to ease the design and debugging process
– writing for subsequent maintenance – version control – regular backup – recognition of relevant social and ethical issues	
• the process of detecting and correcting errors, including: – types of error - syntax errors - logic errors - runtime errors, including: - arithmetic overflow	• differentiate between types of errors • recognise the cause of a specific error and determine how to correct it
- division by zero - accessing inappropriate memory locations	• effectively use a variety of appropriate error correction techniques to locate the cause of a logic error and then correct it
– methods of error detection and correction - use of flags - methodical approach to the isolation of logic errors - use of debugging output statements - peer checking - desk checking - structured walkthrough - comparison of actual with expected output	
• the use of software debugging tools, including: – use of breakpoints – resetting variable contents – program traces – single line stepping	
Documentation of a software solution	
• forms of documentation, including: – log book – user documentation, including:	
- user manual - reference manual - installation guide - tutorial - online help	• produce user documentation (incorporating screen dumps) that includes: – a user manual – a tutorial – online help

Students learn about:	**Students learn to:**
– technical documentation, including: - systems documentation - algorithms - source code	• differentiate between types of user documentation
• use of application software including CASE tools to assist in the documentation process	• identify the personnel who would be likely to use the different types of documentation
• recognition of relevant social and ethical issues	• produce technical documentation for an implemented software solution
Hardware environment to enable implementation of the software solution • hardware requirements – minimum configuration – possible additional hardware – appropriate device drivers or extensions	• recognise the need for additional hardware • identify potential compatibility issues for a newly developed software solution
Emerging technologies • the effect of emerging hardware and software technologies on the development process (see Course Specifications document)	• recognise the implications of emerging technologies for the developer in terms of the code written to make use of these technologies • recognise the implications of emerging technologies for the code development process

9.2.4 Testing and evaluating of software solutions

Students should verify their solutions using test data both at program and system level. Live testing of programs should take place so that potential problems can be identified and addressed. Students should also check that original requirements are met and that there are no logic errors. All user interfaces should also be evaluated at this stage.

These steps are critical in ensuring that the developed product meets the user's needs in terms of relevance, reliability and quality.

Outcomes

A student:

H3.1	identifies and evaluates legal, social and ethical issues in a number of contexts
H3.2	constructs software solutions that address legal, social and ethical issues
H4.2	applies appropriate development methods to solve software problems
H4.3	applies a modular approach to implement well structured software solutions and evaluates their effectiveness
H5.1	applies project management techniques to maximise the productivity of the software development
H5.2	creates and justifies the need for the various types of documentation required for a software solution
H5.3	selects and applies appropriate software to facilitate the design and development of software solutions
H6.1	assesses the skills required in the software development cycle
H6.2	communicates the processes involved in a software solution to an inexperienced user
H6.3	uses and describes a collaborative approach during the software development cycle
H6.4	develops and evaluates effective user interfaces, in consultation with appropriate people.

Students learn about:	Students learn to:
Testing the software solution • comparison of the solution with the design specifications • generating relevant test data for complex solutions • comparison of actual with expected output • levels of testing – module - test that each module and subroutine functions correctly - use of drivers – program - test that the overall program (including incorporated modules and subroutines) functions correctly – system - test that the overall system (including all programs in the suite) functions correctly, including the interfaces between programs - acceptance testing • the use of live test data to ensure that the testing environment accurately reflects the expected environment in which the new system will operate – large file sizes – mix of transaction types – response times – volume of data (load testing) – effect of the new system on the existing systems in the environment into which it will be installed	• differentiate between systems and program test data • test their solution with the test data created at the design stage, comparing actual with expected output • use drivers and/or stubs to test specific modules and subroutines before the rest of the code is developed • recognise the importance of module testing before the module or subroutine is incorporated into the larger solution • recognise that while an individual program or module may have been successfully tested, when it is incorporated into a larger system, problems may become apparent
Reporting on the testing process • documentation of the test data and output produced (see Course Specifications document) – use of CASE tools • communication with those for whom the solution has been developed, including: – test results – comparison with the original design specifications	• demonstrate the features of a new system to the client
Evaluating the software solution • verifying the requirements have been met appropriately • quality assurance	• assess the new software solution to ensure that it meets the specified quality assurance criteria • assess the performance of the new software solution against the criteria specified by the benchmark

Students learn about:	**Students learn to:**
Post implementation review • facilitation of open discussion and evaluation with the client • client sign off process	

9.2.5 Maintaining software solutions

Modifications to source code are often required. Often these are not made by the original developers. Under these circumstances, original documentation is of importance, as is the readability of the source code. As a minimum, all modified or new code should adhere to the standards of the original code.

Students should be given opportunities to modify and document their own code and experience modifying and documenting the code of others. Documentation is an integral part of this process.

Outcomes

A student:

H1.2	differentiates between various methods used to construct software solutions
H3.1	identifies and evaluates legal, social and ethical issues in a number of contexts
H3.2	constructs software solutions that address legal, social and ethical issues
H4.2	applies appropriate development methods to solve software problems
H4.3	applies a modular approach to implement well structured software solutions and evaluates their effectiveness
H5.1	applies project management techniques to maximise the productivity of the software development
H5.2	creates and justifies the need for the various types of documentation required for a software solution
H5.3	selects and applies appropriate software to facilitate the design and development of software solutions
H6.1	assesses the skills required in the software development cycle
H6.2	communicates the processes involved in a software solution to an inexperienced user
H6.3	uses and describes a collaborative approach during the software development cycle
H6.4	develops and evaluates effective user interfaces, in consultation with appropriate people

Students learn about:	**Students learn to:**
Modifying code to meet changed requirements • identifying reasons for change in source code • locating of sections to be altered • determining changes to be made • implementing and testing solution	• read and interpret source code created by other developers • design, implement and test modifications • recognise the cyclical approach to maintenance
Documenting changes • including relevant comments in the source code to highlight the modification • updating associated hard copy documentation and online help • using CASE tools to monitor changes and versions (see Course Specifications document)	• document modifications with dates and reasons for change

9.3 Developing a Solution Package

Project work in the HSC course is intended to reinforce the content covered in the other topics in the course. Students need to experience working collaboratively with their peers and others, as this is common in the computing field beyond school. In order to be able to develop software successfully, students need to be able communicate well with others. Project work gives students these opportunities.

The development of project(s) will build students' understanding of the content dealt with elsewhere in the course and should be integrated throughout the duration of this course.

Outcomes

A student:

H1.1	explains the interrelationship between hardware and software
H1.2	differentiates between various methods used to construct software solutions
H1.3	describes how the major components of a computer system store and manipulate data
H3.1	identifies and evaluates legal, social and ethical issues in a number of contexts
H3.2	constructs software solutions that address legal, social and ethical issues
H4.1	identifies needs to which software solutions are appropriate
H4.2	applies appropriate development methods to solve software problems
H4.3	applies a modular approach to implement well structured software solutions and evaluates their effectiveness
H5.1	applies project management techniques to maximise the productivity of the software development
H5.2	creates and justifies the need for the various types of documentation required for a software solution
H5.3	selects and applies appropriate software to facilitate the design and development of software solutions
H6.1	assesses the skills required in the software development cycle
H6.2	communicates the processes involved in a software solution to an inexperienced user
H6.3	uses and describes a collaborative approach during the software development cycle
H6.4	develops and evaluates effective user interfaces, in consultation with appropriate people

Students learn about:	**Students learn to:**
Designing and developing a software solution to a complex problem	
• defining and understanding the problem	• define the problem and investigate alternative approaches to a software solution
– identification of the problem	
– generation of ideas	• evaluate the ideas for practical implementation
– communication with others involved in the proposed system	• select an appropriate solution
– draft interface design	
– representing the system using diagrams	• produce an initial Gantt chart
– selection of appropriate data structures	
– applying project management techniques	• use a logbook to document the progress of their project (see Course Specifications document)
– consideration of all social and ethical issues	
• planning and designing	• document the software solution
– algorithm design	
– refined systems modeling, such as:	• generate a fully documented design for their project after communication with other potential users
- IPO diagrams	
- context diagrams	
- data flow diagrams (DFDs)	
- storyboards	
- structure charts	
- system flowcharts	• use and modify a Gantt chart as appropriate
- data dictionaries	
– additional resources	
- Gantt charts	
- logbooks	
- algorithms	
- prototypes	
– selecting software environment	
– identifying appropriate hardware	
– selecting appropriate data structures	
– defining files	
- purpose	
- contents	
- organisation	
– defining records	
– defining required validation processes	
– identifying relevant standard or common modules or subroutines	
– using software to document design	
– identifying appropriate test data	
– enabling and incorporating feedback from users at regular intervals	
– considering all social and ethical issues	
– communicating with others involved in the proposed system	
– applying project management techniques	
• implementing	
– converting the solution into code	• implement a fully tested and documented software solution in a methodical manner
– systematic removal of errors	
– refining the data dictionary	
– including standard or common modules or subroutines	• use project management techniques to ensure that the software solution is implemented in an appropriate time frame
– using software to refine documentation	
– creating online help	

Students learn about:	**Students learn to:**
– reporting on the status of the system at regular intervals – applying project management techniques	• ensure that relevant ethical and social issues are addressed appropriately
• testing and evaluating – completing thorough program and system testing – completing all user documentation for the project	• evaluate the project in relation to the original understanding of the problem • review and evaluate the quality of the solution making the necessary changes
• maintaining – modifying the project to ensure: - an improved, more elegant solution - all needs have been met - the software solution operates under changed environments or requirements – updating the software specifications and documentation to reflect the changes	
Whole project issues • project management techniques • social and ethical issues • feedback from users at regular intervals	• manage the project effectively • communicate effectively with potential users

9.4 Options

The option topics in this course extend students' software development experiences in one of two dimensions.

Option 1 Programming Paradigms broadens students' understanding of different types of programming languages by looking at two different types and the reasons for their development.

Option 2 The Interrelationship Between Software and Hardware extends students' understanding of software development by investigating the more detailed relationships between hardware and software and how the hardware is used by the software to allow specified instructions to be performed.

9.4.1 Option 1 Programming Paradigms

This topic offers students the opportunity to look at different types of programming languages. Each of these was developed in an attempt to improve programmer productivity. By focusing on each of the different paradigms, students should gain an insight into how effective each approach has been, together with an understanding of the specific areas where the use of a particular paradigm could be particularly appropriate. This understanding will broaden the students' experience of different paradigms and will also offer them a wider choice from which to select an appropriate language to solve a specific problem.

Students are expected to implement solutions to a number of small relevant problems using an appropriate language. A range of problems should be selected. Some problems will require the use of the logic paradigm, while other problems will require the use of the object oriented paradigm.

Outcomes

A student:

H1.2	differentiates between various methods used to construct software solutions
H2.1	explains the implications of the development of different languages
H2.2	explains the interrelationship between emerging technologies and software development
H4.1	identifies needs to which software solutions are appropriate
H4.2	applies appropriate development methods to solve software problems
H5.3	selects and applies appropriate software to facilitate the design and development of software solutions.

Students learn about:	**Students learn to:**
Development of the different paradigms • limitations of the imperative paradigm – difficulty with solving certain types of problems – the need to specify code for every individual process – difficulty of coding for variability • emerging technologies • simplifying the development and testing of some larger software projects • strengths of different paradigms	• identify the needs that led to the development of different paradigms • recognise the issues associated with using an imperative approach to solve some problems such as Artificial Intelligence (AI) and computer gaming
Logic paradigm • concepts – variables – rules – facts – heuristics – goals – inference engine – backward/forward chaining • language syntax – variables – rules – facts • appropriate use, such as: – pattern matching – AI – expert systems	• recognise representative fragments of code written using the logic paradigm (see Course Specifications document) • recognise the use of the logic paradigm concepts in code • interpret a fragment of code written using the logic paradigm, and identify and correct logic errors • modify fragments of code written using the logic paradigm to incorporate changed requirements • code and test appropriate solutions in a language using the logic paradigm • assess the appropriateness of a software solution written using the logic paradigm against a solution written using an imperative approach
Object oriented paradigm • concepts – classes – objects – attributes – methods/operations – variables and control structures – abstraction – instantiation – inheritance – polymorphism – encapsulation • language syntax – classes – objects – attributes – methods/operations – variables and control structures • appropriate use, such as – computer games – web-based database applications	• recognise representative fragments of code written using the object oriented paradigm (see Course Specifications document) • recognise the use of the object oriented concepts in code • interpret a fragment of code written using the object oriented paradigm, and identify and correct logic errors • modify fragments of code written using the object oriented paradigm to incorporate changed requirements • code and test appropriate solutions in a language using the object oriented paradigm • assess the appropriateness of a software solution written using the object oriented paradigm against a solution written using the imperative approach

Students learn about:	**Students learn to:**
Issues with the selection of an appropriate paradigm • nature of the problem • available resources • efficiency of solution once coded • programmer productivity – learning curve (training required) – use of reusable modules – speed of code generation – approach to testing	• describe the strengths of the imperative, logic and object oriented paradigms • identify an appropriate paradigm relevant for a given situation • evaluate the effectiveness of using a particular paradigm to solve a simple problem

9.4.2 Option 2 The interrelationship between software and hardware

This topic looks in much more depth at how software uses hardware to achieve the desired outcomes. In Section 9.2.3 Implementation of Software Solutions students are introduced to how instructions are processed by the CPU.

In this topic students are shown how data is stored in binary format. Students investigate further how the basic arithmetic processes and storage of data are performed by electronic circuitry. Students should recognise that the design of such circuitry follows the same cyclic process as the design of software – once the problem has been identified, an appropriate solution is designed and tested. A completed circuit can be modified to meet changing requirements and all solutions should be documented and subsequently evaluated.

This topic also introduces students to data streams and their use in communication between the CPU and a range of hardware devices.

Outcomes

A student:

H1.1 explains the interrelationship between hardware and software
H1.3 describes how the major components of a computer system store and manipulate data
H2.2 explains the interrelationship between emerging technologies and software development
H4.1 identifies needs to which software solutions are appropriate
H5.2 creates and justifies the need for the various types of documentation required for a software solution
H5.3 selects and applies appropriate software to facilitate the design and development of software solutions.

Students learn about:	**Students learn to:**
Representation of data within the computer • character representation, namely: – ASCII – Unicode (see Course Specifications document)	• effectively use an ASCII table to convert a character to its equivalent ASCII value and vice versa • recognise the relationship between upper and lower case letters and digits, and their ASCII representation • use the Unicode table which represents a larger character set than is available with ASCII
• representation of data using different number systems – binary – hexadecimal – decimal	• convert a binary or hexadecimal representation to its equivalent character from the ASCII or Unicode table • represent a string of binary digits as its hexadecimal equivalent and vice versa • convert integers between binary, decimal and hexadecimal representations