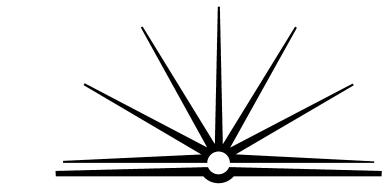


Software Design and Development

Stage 6 Syllabus

Amended 2010

Original published version updated:
September 1999 – Board Bulletin/Official Notices Vol 8 No 7 (BOS 54/99)
June 2009 – Assessment and Reporting information updated
September 2010

Published by
Board of Studies NSW
GPO Box 5300
Sydney NSW 2001
Australia

Tel: (02) 9367 8111
Fax: (02) 9367 8484
Internet: www.boardofstudies.nsw.edu.au

2010630

Contents

1 The Higher School Certificate Program of Study

The purpose of the Higher School Certificate program of study is to:

- provide a curriculum structure which encourages students to complete secondary education;
- foster the intellectual, social and moral development of students, in particular developing their:
  - knowledge, skills, understanding and attitudes in the fields of study they choose
  - capacity to manage their own learning
  - desire to continue learning in formal or informal settings after school
  - capacity to work together with others
  - respect for the cultural diversity of Australian society;
- provide a flexible structure within which students can prepare for:
  - further education and training
  - employment
  - full and active participation as citizens;
- provide formal assessment and certification of students' achievements;
- provide a context within which schools also have the opportunity to foster students' physical and spiritual development.

2 Rationale for Software Design and Development in the Stage 6 Curriculum

For the purposes of the *Software Design and Development Stage 6 Syllabus*, software design and development refers to the creativity, knowledge, values and communication skills required to develop computer programs. The subject provides students with a systematic approach to problem-solving, an opportunity to be creative, excellent career prospects and interesting content.

While a variety of computer applications are used in this subject, they are not the primary focus. The focus of this subject is the development of computer-based solutions that require the design of computer software.

There are many different approaches that can be taken to develop software. An understanding of these and the situations in which they are applied is essential in software development. So too is an understanding of how hardware and software are interrelated and need each other to function. In order to develop solutions that meet the needs of those who will use them, communication, personal and team skills are required by the developers. Together, these considerations provide the basis for the course.

The major focus of the course reflects the traditional structural approach to software development and the top-down development of source code. Although there are other more modern approaches available, the framework of fundamental concepts taught in this course leads to deeper understanding by students, enabling greater flexibility in developing software solutions using newly available technology and languages in the future.

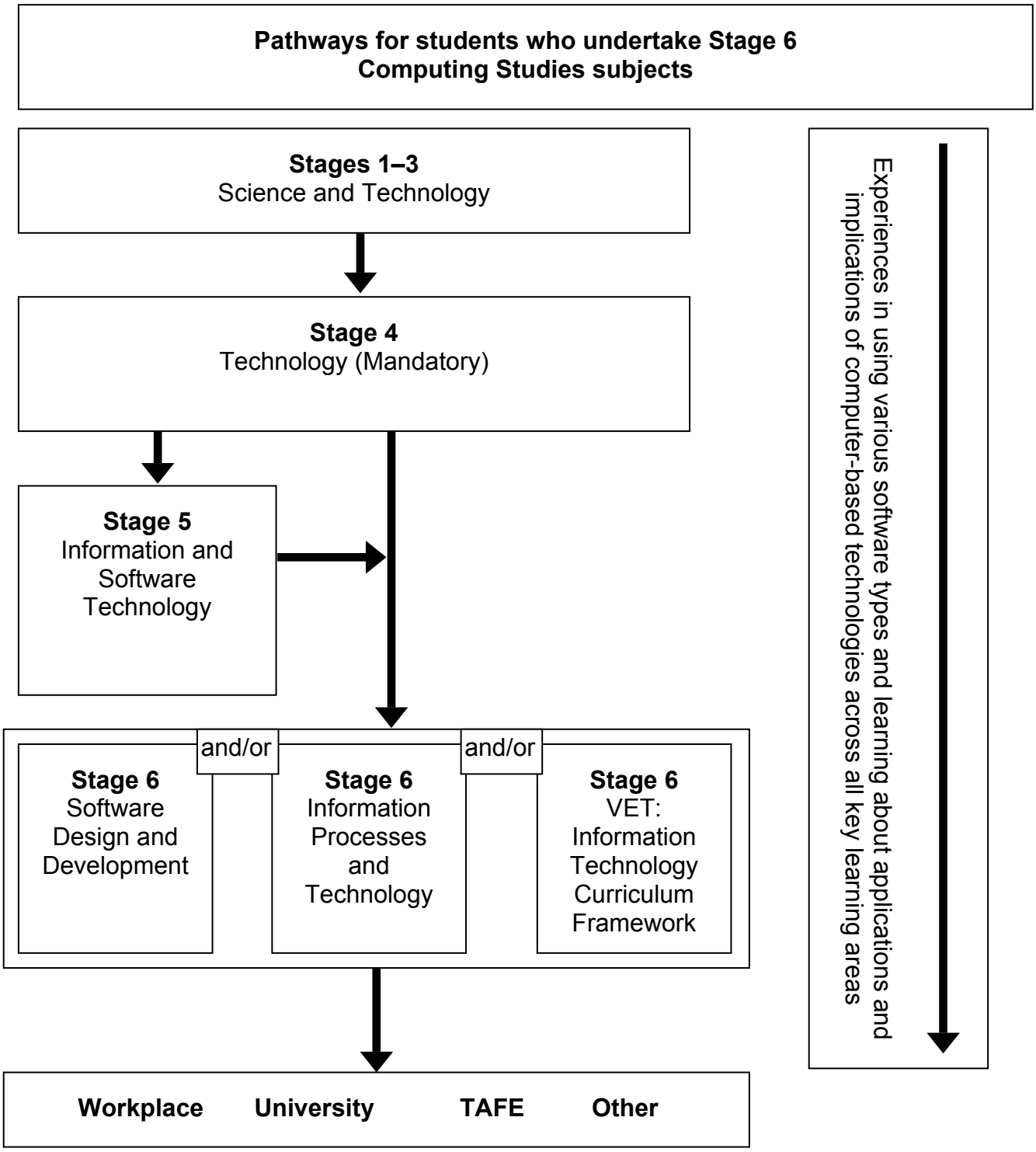
Students interested in the fields of software development and computer science will find this subject of value. The subject is not only for those who seek further study or careers in this field, but also for those who wish to understand the underlying principles of software design and development. Students with software development skills wishing to acquire team and communication skills will find this subject relevant.

The computing field, particularly in the area of software design and development, offers opportunities for creativity and problem-solving and a collaborative work environment where working with people and exploring issues is an integral part of the job. It is critical that students of both genders have the knowledge, understanding and skills necessary to pursue the many new, exciting and highly paid employment opportunities that exist in the field.

The study of Software Design and Development promotes intellectual, social and ethical growth. It provides the flexibility to be able to adapt in a field that is constantly changing, yet vital to the Australian economy.

On completion, the subject provides students with options in the workforce, TAFE and university study. Further, the study of this subject enables students to take part in debates on the suitability, applicability and appropriateness of software solutions to issues in Australian society and the world at large. To this end, Software Design and Development contributes to the overall purpose of the Stage 6 curriculum.

3 Continuum of Learning for Software Design and Development Stage 6 Students



4 Aim

The *Software Design and Development Stage 6 Syllabus* is designed to develop in students the knowledge, understanding, skills and values to solve problems through the creation of software solutions.

5 Objectives

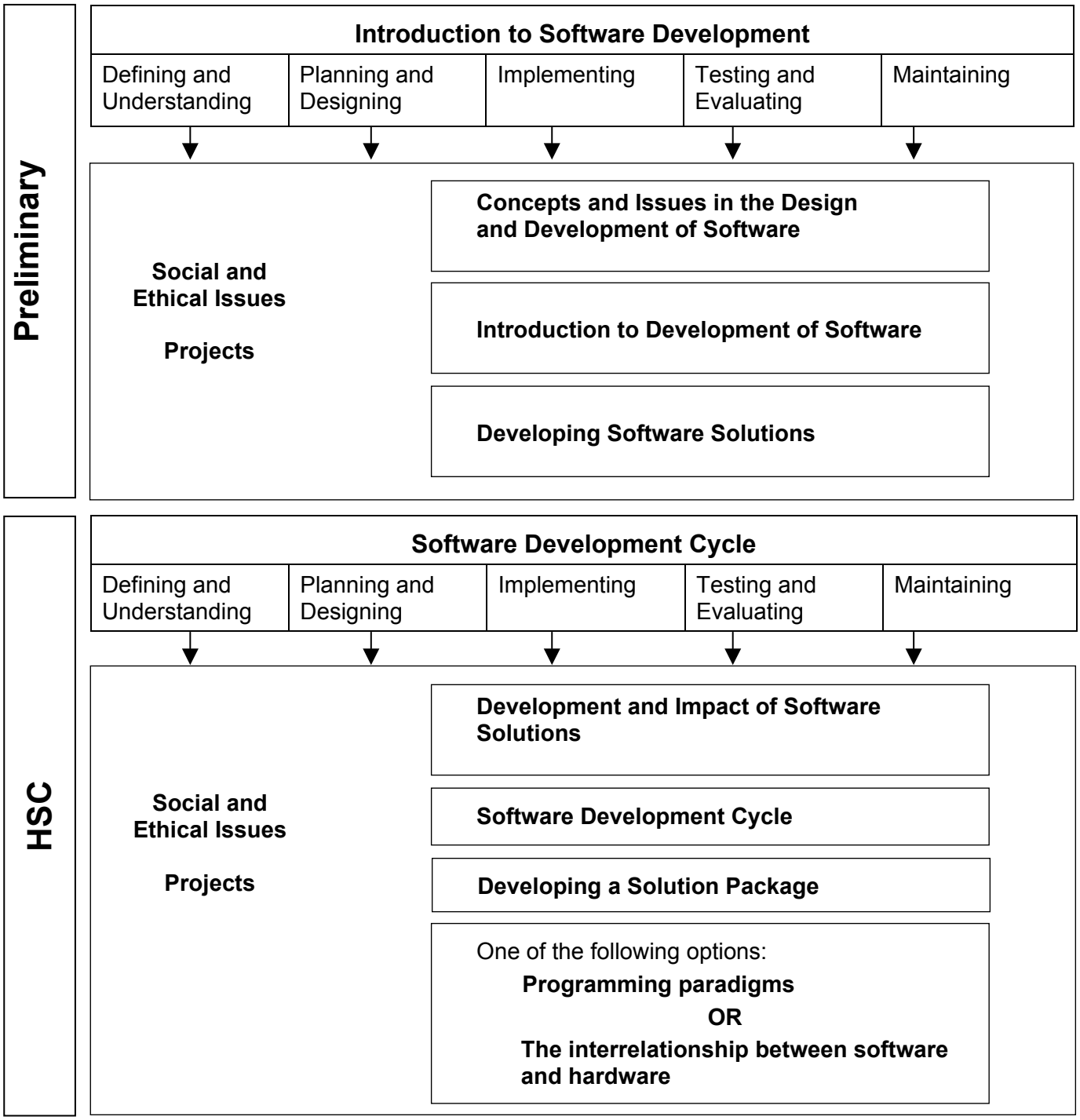
Students will develop:

1. knowledge and understanding about how software solutions utilise and interact with other elements of computer systems
2. knowledge and understanding of the historical developments that have led to current practices in software design and development, and of emerging trends and technologies in this field
3. knowledge and understanding of legal, social and ethical issues and their effect on software design and development
4. skills in designing and developing software solutions
5. skills in management appropriate to the design and development of software solutions
6. skills in teamwork and communication associated with the design and development of software solutions.

6 Course Structure

The following table provides an overview of the arrangement and relationship between components of the Preliminary course and the HSC course for Software Design and Development Stage 6. The percentage values refer to indicative course time.

Preliminary Course **Core strands (100% total time)**	**HSC Course** **Core strands (80% total time)**
Concepts and Issues in the Design and Development of Software 30% • Social and ethical issues • Hardware and software • Software development approaches **Introduction to Software Development 50%** • Defining and understanding the problem • Planning and designing software solutions • Implementing software solutions • Testing and evaluating software solutions • Maintaining software solutions **Developing Software Solutions 20%**	**Development and Impact of Software Solutions 15%** • Social and ethical issues • Application of software development approaches **Software Development Cycle 40%** • Defining and understanding the problem • Planning and designing software solutions • Implementing software solutions • Testing and evaluating software solutions • Maintaining software solutions **Developing a Solution Package 25%** **Options 20%** Study one of the following options: • Programming paradigms OR • The interrelationship between software and hardware


Preliminary
Introduction to Software Development
Defining and Understanding
Planning and Designing
Implementing
Testing and Evaluating
Maintaining
Concepts and Issues in the Design and Development of Software
Social and Ethical Issues
Projects
Introduction to Development of Software
Developing Software Solutions
HSC
Software Development Cycle
Defining and Understanding
Planning and Designing
Implementing
Testing and Evaluating
Maintaining
Development and Impact of Software Solutions
Social and Ethical Issues
Projects
Software Development Cycle
Developing a Solution Package
One of the following options:
Programming paradigms
OR
The interrelationship between software and hardware

7 Objectives and Outcomes

7.1 Table of Objectives and Outcomes

Objectives	**Preliminary outcomes**	**HSC outcomes**
Students will develop: 1. knowledge and understanding about how software solutions utilise and interact with other elements of computer systems	A student: P1.1 describes the functions of hardware and software P1.2 describes and uses appropriate data types P1.3 describes the interactions between the elements of a computer system	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
2. knowledge and understanding of the historical developments that have led to current practices in software design and development, and of emerging trends and technologies in this field	P2.1 describes developments in the levels of programming languages P2.2 describes the effects of program language developments on current practices	H2.1 explains the implications of the development of different languages H2.2 explains the interrelationship between emerging technologies and software development
3. knowledge and understanding of legal, social and ethical issues and their effect on software design and development	P3.1 identifies the issues relating to the use of 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
4. skills in designing and developing software solutions	P4.1 analyses a given problem in order to generate a computer-based solution P4.2 investigates a structured approach in the design and implementation of a software solution P4.3 uses a variety of development approaches to generate software solutions and distinguishes between these approaches	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

Objectives	Preliminary outcomes	HSC outcomes
5. skills in management appropriate to the design and development of software solutions	P5.1 uses and justifies the need for appropriate project management techniques P5.2 uses and develops documentation to communicate software solutions to others	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
6. skills in teamwork and communication associated with the design and development of software solutions	P6.1 describes the skills involved in software development P6.2 communicates with appropriate personnel throughout the software development process P6.3 designs and constructs software solutions with appropriate interfaces	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

7.2 Key Competencies

Software Design and Development provides a context within which to develop general competencies considered essential for the acquisition of effective, higher-order thinking skills necessary for further education, work and everyday life.

The key competencies are explicitly addressed in the Software Design and Development syllabus to enhance student learning. The key competency of ***collecting, analysing and organising information*** is addressed through the planning stage, when students are required to determine what the problem is and how it may best be solved.

Communicating ideas and information is a skill developed by students so that they can both understand the nature of the problem to be solved and ensure that the proposed solution meets the users' needs.

Planning and organising activities and working with others and in teams are integral to the development of software and are addressed in Preliminary and HSC courses, mainly through the development of software solutions using effective project management techniques.

Using mathematical ideas and techniques is addressed as students formulate algorithms, investigate data structures with consideration to how they are presented internally, and construct timelines or analyse statistical evidence.

During investigations, students will need to select and use appropriate information technologies, thereby developing the key competency of ***using technology***.

Finally, the exploration of issues and investigation and solution of problems contributes towards the students' development of the key competency ***solving problems***.

8 Content: Software Design and Development Stage 6 Preliminary Course

8.1 Concepts and Issues in the Design and Development of Software

8.1.1 Social and ethical issues

Students are introduced to the widespread use of software in society and are asked to consider how the software originated and what issues needed to be considered during its development. One way in which students can achieve this is to reflect on a range of existing software applications.

This topic identifies social and ethical issues that arise in the development and use of software. Students should be made aware of these issues early in the course so that they can act in a socially responsible and ethical way throughout the course. Although these issues are taught specifically as part of this topic, they should also be reconsidered as each new topic is discussed. Thus, for example, interface design issues, duplication of code or ideas and language used in documentation should all be considered again at relevant parts in the course.

Outcomes

A student:
P2.2 describes the effects of program language developments on current practices
P3.1 identifies the issues relating to the use of software solutions
P6.1 describes the skills involved in software development.

Students learn about:	Students learn to:
Evolution of software applications • significant applications and design features such as: – Command line interface – GUI interface – search engines – VisiCalc – web browsers – presentation software – email – social networking applications	• identify significant milestones in the evolution of software applications and design features
Intellectual property • copyright • types of software licences • licence terminology • legal aspects • use of software covered by a licence agreement (see Course Specifications document) • events that have led to the need for software licence agreements, including: – ease of reproduction and copy – collaborative development history – the current open environment of the internet • sources of code and license conditions that apply, such as: – the internet – books and magazines	• analyse the issues relating to intellectual property • appropriately acknowledge externally sourced code • use software in an ethically and legally correct manner

Students learn about:	Students learn to:
Social context of software design ***Ergonomics*** • ergonomic issues regarding software design: – effectiveness of screen design – ease of use – appropriate messages to the user – consistency of the user interface	
Inclusivity • the need for software to not exclude individuals or groups based on characteristics such as: – cultural background – economic background – gender – disability	• design and evaluate software interfaces in terms of inclusivity
Privacy • need to protect an individual's data and identity	• identify ways in which privacy can be protected
Required skills in software design and development, including: • communication skills • ability to work in teams • creativity • design skills • technical skills • problem-solving skills • attention to detail	• identify the range of skills required to complete a minor software project

8.1.2 Hardware and software

Hardware and software are mutually dependent components of a computer system. To fully appreciate their role in a computer system they should be examined in conjunction with data, processes and personnel.

This topic provides students with a holistic understanding of a computer system and its role in software development.

Outcomes

A student:

P1.1 describes the functions of hardware and software

P1.3 describes the interactions between the elements of a computer system

P2.1 describes developments in the levels of programming languages

P2.2 describes the effects of program language developments on current practices

P3.1 identifies the issues relating to the use of software solutions

P6.1 describes the skills involved in software development.

Students learn about:	Students learn to:
Elements of a computer system • hardware • software • data • procedures • personnel	• identify the elements of a computer system and their role in that system • describe the significance of and interaction between the elements comprising computer systems
Hardware • the function of hardware within a computer system, namely: – input – output – process – storage – control • how a variety of input devices, output devices, storage devices and CPU components achieve their purpose	• describe how data is captured, stored, manipulated or displayed on a variety of hardware devices (see Course Specifications document) • competently use computer hardware, selecting appropriate hardware for specific tasks
• the current trends and developments in computer hardware	• identify the impact of using particular devices on the development and use of software
Software • operating systems and utilities (see Course Specifications document) • off-the-shelf applications packages and custom-designed software	
• generations of programming languages, namely: – machine code: 1st generation – assembly language: 2nd generation – higher-level languages (imperative/procedural): 3rd generation – declarative (non-procedural) languages: 4th generation	• competently use a range of software • describe the development of the generations of programming languages • identify the effect of the generations of programming languages on software development practices
• the need for translation – compilation	• distinguish between methods of translation

Students learn about:	**Students learn to:**
– interpretation • functions of operating systems – provide interface to hardware – provide interface to user – provide interface to software applications – control the concurrent running of software applications – manage system resources (see Course Specifications document) • current trends in the development of software and operating systems **The relationship between hardware and software** • processing of software instructions by hardware – the fetch–execute cycle • the initiation and running of an application by the operating system – locate and load application – hand control to application – start fetch–execute cycle for the application • the existence of minimum hardware requirements to run some software	• identify typical tasks performed by operating systems • describe what happens during each of the steps of the fetch–execute cycle • identify the role of specific hardware used during each step of the fetch–execute cycle

8.1.3 Software development approaches

There are a number of different approaches that can be taken when developing software. Five are prescribed for study in this course. There are many ways in which software is commercially developed, from an ad-hoc approach to the more formalised structured approach. The approach or combination of approaches used depends on the nature of the problem to be solved, the resources available and the experience of those involved. This topic introduces students to some of the alternative approaches and the relevance of each.

Outcomes

A student:
P2.2 describes the effects of program language developments on current practices
P3.1 identifies the issues relating to the use of software solutions
P4.1 analyses a given problem in order to generate a computer-based solution
P4.2 investigates a structured approach in the design and implementation of a software solution
P4.3 uses a variety of development approaches to generate software solutions and distinguishes between these approaches
P6.1 describes the skills involved in software development.

Students learn about:	**Students learn to:**
Structured approach • stages in program development – defining and understanding the problem – planning and designing – implementing – testing and evaluating – maintaining • characteristics of the structured approach, including: – distinct formal stages – long time periods – large-scale projects – large budgets – involvement of a development team consisting of: - analysts - designers - programmers - clients (users and management)	• identify each of these stages in practical programming exercises
Agile approach • speed of getting solution to market • interactive approach with selective refinement • working version delivered after each iteration • responds well to changing specifications • close collaboration between development team and client throughout the development process	
Prototyping • modeling of a proposed solution or part of a solution • progressive refinement of the model in response to feedback	• design and develop a limited prototype as a proposed solution, or part of a solution, to a problem

Students learn about:	**Students learn to:**
Rapid applications development approach (RAD) • characteristics of the rapid approach, including: – lack of formal stages – use of existing routines – use of appropriate applications to develop the RAD solution - drag and drop programming environments - common application packages such as spreadsheets, databases – communication between developer and client – short time period – small-scale projects – small budgets	• analyse the effectiveness of the prototyping approach in developing a software solution • use an existing software package to develop a solution using a RAD approach
End user approach • characteristics of the end user approach, including: – end user as the developer and maintainer – typically uses RAD and/or prototyping – the developer is the client, therefore there are no communication issues – small budget and/or short time period for development	• discuss the advantages and disadvantages of end user developed software
Selecting an appropriate development approach • software solutions that have been developed using a single approach • software solutions that have been developed using a combination of approaches	• compare and contrast structured and agile approaches • recognise reasons for the failure of solutions • select appropriate software development approaches for specific purposes • identify characteristics of projects that lend themselves to a specific development approach • recognise that a single solution may involve a combination of approaches • identify characteristics of projects that require a combination of approaches

8.2 Introduction to Software Development

All software development approaches include the phases of defining and understanding the problem, planning and designing, implementing, testing, and evaluating and maintaining. There are variations in the time, sequence and organisation of these phases in each of the approaches introduced in this course. Students may use more than one approach in this course. The content for each of the phases is listed below and should be presented to students in a cyclic fashion. Areas for investigation could include writing structured code, modeling and simulation, scripting hypermedia tools, and customisation of application packages through modifying or creating scripts.

It is important that these areas of investigation involve the use of data types, control structures and other content covered in this unit.

8.2.1 Defining and understanding the problem, and planning and designing software solutions

In planning a solution, students need to understand the problem to be solved and how the solution will be used. In this topic, students will consider all aspects of the solution before starting its implementation. The selection of data types and structures used in the solution of a problem can have a huge impact on the effectiveness of that solution. A variety of data types and structures are introduced in this topic and appropriate algorithms should be developed and implemented that make best use of these. As algorithms become more complex, there is a need for a methodical top-down approach with progressive refinement of detail. It is important that algorithms use the control structures as specified in Course and Software Specifications document. Problems should be selected at a level of difficulty commensurate with the ability level of students.

Outcomes

A student:

P1.2 describes and uses appropriate data types

P1.3 describes the interactions between the elements of a computer system

P2.2 describes the effects of program language developments on current practices

P3.1 identifies the issues relating to the use of software solutions

P4.2 investigates a structured approach in the design and implementation of a software solution

P4.3 uses a variety of development approaches to generate software solutions and distinguishes between these approaches

P5.2 uses and develops documentation to communicate software solutions to others.

Students learn about:	**Students learn to:**
Understanding the problem • clarification of the specifications • performance requirements • identification of inputs and required outputs • determining the steps that, when carried out, will solve the problem • Input Process Output (IPO) diagrams	• determine the inputs and outputs required for a particular problem • produce an IPO diagram from a set of specifications
Abstraction/refinement • the top-down approach to solution development – a system comprises all the programs in the suite – a program comprises all of the modules required to perform the required task – a module is a group of subroutines that together achieve a subtask – a subroutine is a set of statements that performs a single logical task	• develop a systematic approach to the development of software solutions • document a proposed non-complex software solution – represent the flow of data through a system using a context diagram – represent a system using a data flow diagram (DFD) to show its components and the data transferred between them – represent a system using a structure chart to show the interrelationship between the component modules – represent a system using a systems flowchart to show its component modules, files and media
Data types • data types used in solutions, including: – integer – string – floating point/real – boolean	
• integer representation in binary, decimal and hexadecimal • characters represented as numbers in binary, decimal and hexadecimal • limitations of particular data types • data structures, including: – one-dimensional array – record • use of records in sequential files	• interpret and use an ASCII table • identify the maximum decimal value that can be stored in a given number of bits • recognise the impact of the use of an inappropriate data type • select the most appropriate data type for the solution to a particular problem and discuss the merit of the chosen type • create a data dictionary which defines the data appropriately

Students learn about:	**Students learn to:**
Structured algorithms • control structures which form the basic building blocks of all algorithms: – sequence – selection (binary, multiway) – repetition (pre-test, post-test), including for … next loops – use of subroutines	• identify control structures in an algorithm
• methods for representing algorithms: – pseudocode – flowcharts incorporating standard control structures • software structure – use of a clear uncluttered mainline and subroutines – use of a modular approach – use of stubs to represent incomplete modules • use of standard algorithms, including: – load an array and print its contents – add the contents of an array of numbers	• interpret and create algorithms represented in both pseudocode and flowcharts that use standard control structures
• checking the algorithm for errors	• detect logic errors in an algorithm by performing a desk check
• benefits of using structured algorithms – ease of development – ease of understanding – ease of modification	• gather solutions from a number of sources and modify them to form an appropriate solution to a specified problem
	• represent code from different sources as an algorithm to assist in understanding its purpose and to assess its relevance in a proposed solution
	• incorporate a stub for modules for which the detail has not yet been developed

8.2.2 Implementing software solutions

The implementation phase could involve a range of activities from modifying existing code to the development of new code. In order to implement a solution, students need to understand the syntax of the chosen language.

Careful consideration needs to be given to the language used to implement solutions. The chosen language should be one that best reinforces the concepts being taught, not simply one that is currently fashionable. In some cases, this may be a scripting language for an applications package. It is recognised that in a school environment, the choice of language may well be limited by the skills and resources available. It is important, however, that any language used meets the course requirements as specified in Course and Software Specifications.

Regardless of the language used, students should be familiar with using EBNF or railroad diagrams that specify the valid syntax of the commands used. For every set of algorithms that is implemented, appropriate user interfaces will need to be developed along with suitable documentation. Relevant social and ethical issues should be addressed, particularly with reference to appropriate interface design and issues related to using third party designs and code.

Outcomes

A student:

P1.2 describes and uses appropriate data types

P1.3 describes the interactions between the elements of a computer system

P3.1 identifies the issues relating to the use of software solutions

P4.2 investigates a structured approach in the design and implementation of a software solution

P4.3 uses a variety of development approaches to generate software solutions and distinguishes between these approaches

P5.2 uses and develops documentation to communicate software solutions to others

P6.1 describes the skills involved in software development

P6.2 communicates with appropriate personnel throughout the software development process

P6.3 designs and constructs software solutions with appropriate interfaces.

Students learn about:	**Students learn to:**
Coding in an approved programming language • meta-languages, including: – EBNF – railroad diagrams • language syntax – specified through meta-languages in manuals and help documentation	• verify the syntax of a command using meta-language statements
• the syntax used to represent the control structures, including: – sequence – selection (binary, multiway) – repetition (pre-test, post-test, for…next loops) – use of subroutines or procedures – combinations of these	• specify syntax using meta-language statements • use meta-language statements to develop syntactically correct code
• the syntax used to define and use a range of data types and data structures, including: – integer – string – floating point/real – boolean – one-dimensional array – records	• generate appropriate source code by: – using appropriate data types and data structures in solutions – using a programming environment to generate and execute code – coding an algorithm into the chosen programming language
Developing source code – converting algorithms into source code using syntactically correct statements	
Error detection and correction techniques • types of coding errors, including: – syntax errors – runtime errors – logic errors • stubs – used to check the flow of execution – used to replace subroutines/modules during testing to check if that section of the code is the cause of an error • flags – used to check if a section of code has been executed – can be used as part of the logic of a solution or as an error detection process • debugging output statements – additional print statements in the code for use in the debugging process – used to identify which sections of the code have been executed – used to interrogate variable contents at a particular point in the execution of a program	• trace the output of a given code fragment and modify it appropriately • systematically eliminate syntax errors so that a program can be executed • run, correct and extend existing code • test a program with boundary values to detect possible runtime errors • detect and correct logic errors in program code by using a systematic error detection and correction process

Students learn about:	**Students learn to:**
Commonly executed sections of code • reusable code – standard logic, such as: - a login process - data validation - conversion between date formats – to replace multiple occurrences of the same code • combining code from different sources – copying and pasting into code – calling modules or subroutines • making the same data available to different modules – global variables – parameter passing • use of functions and procedures	• develop standard modules or subroutines for reuse • create solutions to problems using existing code with minimal change or additions • develop code that makes use of common modules or subroutines • differentiate between the scope of local and global variables • develop code that makes appropriate use of global and local variables • develop code that calls common modules and passes parameters appropriately • incorporate functions into modules or subroutines • make use of procedures (see Course Specifications document)
User interface development • the need for consultation with users and/ or managers • use of storyboard – shows the general design of each interface – shows navigation between interfaces • effective user interfaces, including: – factors affecting readability – use of white space – effective prompts – judicious use of colour and graphics – grouping of information – unambiguous and non-threatening error messages – legibility of text, including: - justification - font type (serif vs sans serif) - font size - font style - text colour – navigation – recognition of relevant social and ethical issues – consistency – appropriate language for the intended audience	• develop solutions that include appropriate user interfaces • evaluate the effectiveness of interfaces used in commercially available software • develop an appropriate storyboard for a specified problem • design screens incorporating good design and ergonomic features • incorporate current relevant interface elements into software solutions
Documentation • types of documentation – documentation for developers – documentation for users • internal documentation – meaningful variable names (intrinsic) – readability of code	• produce documentation for different audiences • produce source code which is well documented and therefore easy to read, understand and maintain • fully document a solution that has been developed in the classroom

Students learn about:	**Students learn to:**
- comments - white space - indentation • online help, such as: – context sensitive help – help files	• create a data dictionary to define the data (including variables, arrays and records) used in a developed solution • use a range of application packages to develop the various types of documentation to fully document a solution • interpret code and documentation prepared by others • assess the effectiveness of online help available in software packages

8.2.3 Testing and evaluating software solutions

Students should check their code using test data that test their programs thoroughly. Students should check that their solution meets the required objectives. Specifications for a problem together with the coded solution to that problem should be given to students and they should be asked to test the solution to see if it meets the specifications. It is important for students to recognise the responsibilities of software developers in terms of providing a software solution that is appropriate to the defined problem and that runs effectively under all possible conditions. Developed software must be thoroughly tested to ensure that it will not fail unexpectedly or produce irrelevant results even when exposed to unusual or unexpected conditions.

It should be noted that students are expected to have tested their partially coded solutions at various stages throughout its development.

Outcomes

A student:

P3.1 identifies the issues relating to the use of software solutions

P4.2 investigates a structured approach in the design and implementation of a software solution

P5.1 uses and justifies the need for appropriate project management techniques

P5.2 uses and develops documentation to communicate software solutions to others

P6.1 describes the skills involved in software development

P6.2 communicates with appropriate personnel throughout the software development process

P6.3 designs and constructs software solutions with appropriate interfaces.

Students learn about:	Students learn to:
Testing the solution • the selection of appropriate test data, including: – data that test all the pathways through the algorithm – data that test boundary conditions ‘at’, ‘above’ and ‘below’ values upon which decisions are based – data where the required answer is known – data which is outside the expected values • the need for thorough test data • testing both algorithms and coded solutions with test data such as: – desk checking an algorithm – stepping through a coded solution line by line • peer checking • structured walk through	• determine the expected result given test data • compare the actual output from a piece of code with the expected output from test data to detect logic errors • create a set of appropriate test data and use it to verify the logic in a solution • perform a desk check by producing a table showing the changes to the content of variables as the algorithm or code is stepped through manually
Evaluating the solution • comparing different solutions to the same problem – different interpretations of the design specifications – the advantages and disadvantages of different approaches to a solution	• critically evaluate their work and that of their peers • share good aspects of their solutions and the solutions of others

Students learn about:	**Students learn to:**
• checking the solution to see if it meets the original design specifications • the importance and use of user feedback • the importance of checking that social and ethical perspectives have been appropriately addressed	

8.2.4 Maintaining software solutions

Modifications to code are often required. These modifications need not be made by the original developers. In these situations, original documentation is very important in understanding the logic used in the solution. Students should be given opportunities to modify their code and to gain experience in modifying the code of others with varying amounts of documentation available. Students could be asked to modify solutions as a means of assessing their understanding. Students should be reminded of the ethical issues associated with accessing and modifying the code of others.

Outcomes

A student:

P1.2 describes and uses appropriate data types

P2.2 describes the effects of program language developments on current practices

P3.1 identifies the issues relating to the use of software solutions

P4.1 analyses a given problem in order to generate a computer-based solution

P4.2 investigates a structured approach in the design and implementation of a software solution

P4.3 uses a variety of development approaches to generate software solutions and distinguishes between these approaches

P5.1 uses and justifies the need for appropriate project management techniques

P5.2 uses and develops documentation to communicate software solutions to others

P6.1 describes the skills involved in software development

P6.2 communicates with appropriate personnel throughout the software development process

P6.3 designs and constructs software solutions with appropriate interfaces.

Students learn about:	Students learn to:
Reasons for maintaining code • changing user requirements • upgrading the user interface • changes in the data to be processed • introduction of new hardware or software • changing organisational focus • changes in government requirements • poorly implemented code	• identify and describe features in code that allow it to be easily maintained
Features in source code that improve its maintainability, including: • use of variables instead of literal constants • use of meaningful variable names • explanatory comments in the code • use of standard control structures with appropriate indentation • appropriate use of white space to improve legibility of the source code • a clear and uncluttered mainline • one logical task per subroutine • meaningful names for subroutines and modules	• create solutions that are easy to maintain

Students learn about:	**Students learn to:**
Understanding source code • reading original documentation in order to understand code – documentation for the user (including user manuals) – documentation for developers • reading original algorithms to identify: – inputs – the type and purpose of variables used – processes – outputs • creating algorithms for source code when they are not available to aid in understanding – identify the control structures that have been used – understand how variables have been used	• convert a fragment of source code into its equivalent algorithm • define the purpose of the code to be maintained • modify code to meet changed requirements
Inclusion of code from other sources • copyright issues • compatibility of code	• provide appropriate acknowledgement of the code of other programmers that has been incorporated as part of the maintenance process • assess the compatibility of code to be included in the source code of an existing solution

8.3 Developing Software Solutions

A series of programming tasks allow the students to put into practice the concepts covered in the Preliminary course. They allow students to build solutions from specifications and to apply appropriate project management techniques.

Working in teams is common in the computing field. In order to be a successful member of a team, students need to communicate well with others and to act in a social and ethical way.

In this topic, students can work with others to develop software solutions. Students should ensure that their solutions appropriately address all relevant social and ethical issues.

Outcomes

A student:

P1.2 describes and uses appropriate data types
P1.3 describes the interactions between the elements of a computer system
P3.1 identifies the issues relating to the use of software solutions
P4.1 analyses a given problem in order to generate a computer-based solution
P4.2 investigates a structured approach in the design and implementation of a software solution
P4.3 uses a variety of development approaches to generate software solutions and distinguishes between these approaches
P5.1 uses and justifies the need for appropriate project management techniques
P5.2 uses and develops documentation to communicate software solutions to others
P6.2 communicates with appropriate personnel throughout the software development process
P6.3 designs and constructs software solutions with appropriate interfaces.

Students learn about:	Students learn to:
Project management • identifying tasks • identifying required programs, modules and subroutines • Gantt charts • logbooks – regular record of progress – record of major milestones and stumbling blocks • allocating resources • regular backup with version numbers • responding to difficulties – reference to documentation such as manuals – discussion with peers and experts – reporting problems to management • evaluating the solution – throughout the process – on completion	• use appropriate project management techniques • create and use Gantt charts and logbooks • devise, document and implement an appropriate backup strategy that incorporates relevant version numbers
Documenting software solutions • IPO diagrams • context diagrams • data flow diagrams (DFDs) • storyboards • structure charts	• prepare suitable documentation to accompany software solutions • use appropriate application packages in creating documentation to support a software solution

Students learn about:	**Students learn to:**
• system flowcharts • data dictionaries • Gantt charts • logbooks • algorithms • user documentation including manuals and online help	• create appropriate systems documentation for a variety of programming tasks
Developing software solutions • defining and understanding the problem – preparation of initial documentation • planning and designing – identification of a suitable development approach – design of appropriate algorithms – identification and incorporation of appropriate existing algorithms – determination of appropriate data structures – identification of relevant subroutines – design of test data and expected output – desk check of algorithms – identification of existing code that can be used • implementing – coding the solution in an appropriate language – testing using test data – documenting the solution, including: - algorithms - test data and expected output - data dictionary - user documentation	• apply the steps in the software development cycle when developing solutions • produce a working solution from an algorithm derived from a set of specifications
• testing and evaluating – testing of the solution using test data – evaluating the implemented solution	• effectively test a solution
• maintaining – modifying the solution to meet original or changed specifications	• update a solution incorporating new requirements
Social and ethical issues related to software solutions • intellectual property • ergonomics issues • inclusivity and accessibility • privacy	• address relevant social and ethical issues in their software solutions

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) **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	• 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 • 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 • execution of called routines • linking, including use of DLLs	• recognise, interpret and write machine code instructions for a problem fragment
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 – writing for subsequent maintenance – version control – regular backup – recognition of relevant social and ethical issues	• 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
• the process of detecting and correcting errors, including: – types of error - syntax errors - logic errors - runtime errors, including: - arithmetic overflow - division by zero - accessing inappropriate memory locations – 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	• differentiate between types of errors • recognise the cause of a specific error and determine how to correct it • effectively use a variety of appropriate error correction techniques to locate the cause of a logic error and then correct it
• 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 • use of application software including CASE tools to assist in the documentation process • recognition of relevant social and ethical issues **Hardware environment to enable implementation of the software solution** • hardware requirements – minimum configuration – possible additional hardware – appropriate device drivers or extensions **Emerging technologies** • the effect of emerging hardware and software technologies on the development process (see Course Specifications document)	• differentiate between types of user documentation • identify the personnel who would be likely to use the different types of documentation • produce technical documentation for an implemented software solution • recognise the need for additional hardware • identify potential compatibility issues for a newly developed software solution • 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 – identification of the problem – generation of ideas – communication with others involved in the proposed system – draft interface design – representing the system using diagrams – selection of appropriate data structures – applying project management techniques – consideration of all social and ethical issues • planning and designing – algorithm design – refined systems modeling, such as: - IPO diagrams - context diagrams - data flow diagrams (DFDs) - storyboards - structure charts - system flowcharts - 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 – systematic removal of errors – refining the data dictionary – including standard or common modules or subroutines – using software to refine documentation – creating online help	• define the problem and investigate alternative approaches to a software solution • evaluate the ideas for practical implementation • select an appropriate solution • produce an initial Gantt chart • use a logbook to document the progress of their project (see Course Specifications document) • document the software solution • generate a fully documented design for their project after communication with other potential users • use and modify a Gantt chart as appropriate • implement a fully tested and documented software solution in a methodical manner • use project management techniques to ensure that the software solution is implemented in an appropriate time frame

Students learn about:	**Students learn to:**
– reporting on the status of the system at regular intervals – applying project management techniques • testing and evaluating – completing thorough program and system testing – completing all user documentation for the project • 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	• ensure that relevant ethical and social issues are addressed appropriately • evaluate the project in relation to the original understanding of the problem • review and evaluate the quality of the solution making the necessary 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

Students learn about:	**Students learn to:**
• integer representation, including: – sign and modulus – 1's complement – 2's complement	
• floating point/real representation – very large positive and negative values – very small positive and negative values – integer and non-integer values – limitations	• convert between decimal fractions and the equivalent IEEE754 single precision floating point representation • recognise implications of the limitations of particular data representations
• binary arithmetic, including: – addition – subtraction using 2's complement representation – multiplication (shift and add) – division (shift and subtract)	• perform arithmetic operations in binary
Electronic circuits to perform standard software operations	
• logic gates, including: – AND, OR, NOT, NAND, NOR, XOR • truth tables • Boolean algebra – describing a circuit – simplifying an existing circuit • circuit design steps – identify inputs and outputs – identify required components – check solution with a truth table – evaluate the circuit design • specialty circuits, including: – half adder – full adder – flip-flops	• generate truth tables for a given circuit • describe the function of a circuit from its truth table • design a circuit to solve a given problem • convert between the Boolean representation of a circuit and its circuit diagram • build and test both user-designed and specialty circuits using integrated circuits or simulation software • use a cyclical approach when designing circuits • modify an existing circuit design to reflect changed requirements • describe the function of specialty circuits • analyse a specialty circuit in order to determine its output • explain how a flip-flop can be used in the storage and shifting of a bit in memory
Programming of hardware devices	
• the data stream – format of the data stream - header information - data block - trailer information – use of control characters – use of hardware specifications to describe the expected format of the data stream	• interpret a data stream for a device for which specifications are provided • modify a stream of data to meet changed requirements, given the hardware specifications • generate a data stream to specify particular operations for a hardware device, for which specifications are provided such as a printer, to specify line feed, form feed, font and style change, and line spacing

Students learn about:	**Students learn to:**
• processing an input data stream from sensors and other devices – the structure of the data stream - the need to recognise and strip control characters - the need to identify the data characters – interpreting the data stream (see Course Specifications document) • generating output to an appropriate device – determining the purpose of the output – the structure of the data stream - required header information - the need for control characters - specification of data characters - required trailer information (see Course Specifications document)	• develop an algorithm to identify and extract data and/or control characters in order to interpret a data stream sent from the hardware • develop an algorithm to generate a data stream to provide relevant instructions to the hardware
• issues with interpreting data streams	• recognise that a string of binary digits can have many different meanings • interpret a string of binary digits, given a number of different possible specifications

10 Course Requirements

The Software Design and Development Stage 6 Syllabus includes a Preliminary course of 120 hours (indicative time) and an HSC course of 120 hours (indicative time).

There is no prerequisite study for the Preliminary course. Completion of the Preliminary course is a prerequisite for the HSC course.

It is a mandatory requirement that students spend a minimum of 20% of Preliminary course time and 25% of HSC course time on practical activities using the computer.

Software Specifications and Methods of Algorithm descriptions prescribed for Software Design and Development Stage 6

There are Software Specifications and Methods of Algorithm descriptions prescribed for Software Design and Development Stage 6 Preliminary and HSC courses. These are published in the Board Bulletin on the Board of Studies website (www.boardofstudies.nsw.edu.au).

11 Post-school Opportunities

The study of Software Design and Development Stage 6 provides students with knowledge, understanding and skills that form a valuable foundation for a range of courses at university and other tertiary institutions.

In addition, the study of Software Design and Development Stage 6 assists students to prepare for employment and full and active participation as citizens. In particular, there are opportunities for students to gain recognition in vocational education and training. Teachers and students should be aware of these opportunities.

Recognition of Student Achievement in Vocational Education and Training (VET)

Wherever appropriate, the skills and knowledge acquired by students in their study of HSC courses should be recognised by industry and training organisations. Recognition of student achievement means that students who have satisfactorily completed HSC courses will not be required to repeat their learning in courses at TAFE NSW or other Registered Training Organisations (RTOs).

Registered Training Organisations, such as TAFE NSW, provide industry training and issue qualifications within the Australian Qualifications Framework (AQF).

The degree of recognition available to students in each subject is based on the similarity of outcomes between HSC courses and industry training packages endorsed within the Australian Qualifications Framework. Training packages are documents that link an industry's competency standards to AQF qualifications. More information about industry training packages can be found on the National Training Information Service (NTIS) website (www.ntis.gov.au).

Recognition by TAFE NSW

TAFE NSW conducts courses in a wide range of industry areas, as outlined each year in the TAFE NSW Handbook. Under current arrangements, the recognition available to students of Software Design and Development in relevant courses conducted by TAFE is described in the HSC/TAFE Credit Transfer Guide. This guide is produced by the Board of Studies and TAFE NSW and is distributed annually to all schools and colleges. Teachers should refer to this guide and be aware of the recognition available to their students through the study of Software Design and Development Stage 6. This information can be found on the TAFE NSW website (www.tafensw.edu.au/mchoice).

Recognition by other Registered Training Organisations

Students may also negotiate recognition into a training package qualification with another RTO. Each student will need to provide the RTO with evidence of satisfactory achievement in Software Design and Development Stage 6 so that the degree of recognition available can be determined.

12 Assessment and Reporting

Advice on appropriate assessment practice in relation to the Software Design and Development syllabus is contained in *Assessment and Reporting in Software Design and Development Stage 6*. That document provides general advice on assessment in Stage 6 as well as the specific requirements for the Preliminary and HSC courses. The document contains:

- suggested components and weightings for the internal assessment of the Preliminary course
- mandatory components and weightings for the internal assessment of the HSC course
- the HSC examination specifications, which describe the format of the external HSC examination.

This document and other resources and advice related to assessment in Stage 6 Software Design and Development are available on the Board's website at www.boardofstudies.nsw.edu.au/syllabus_hsc