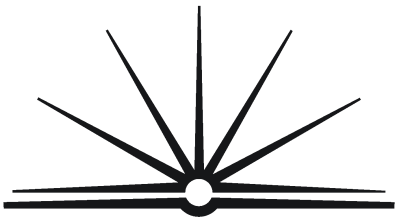
Tip: Click on the BOOKMARKS TAB to show / hide navigation



BOARD OF STUDIES
NEW SOUTH WALES

This syllabus is current as of January 2003. The Office of the Board of Studies NSW website may have a more current version available to download.

www.boardofstudies.nsw.edu.au

Software Design and Development

Stage 6

Syllabus

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

Tel: (02) 9367 8111

Internet: http://www.boardofstudies.nsw.edu.au

ISBN 0 7313 4290 9

99278

Contents

1 The Higher School Certificate Program of Study 5

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

3 Continuum of Learning for Software Design and Development Stage 6 Students...7

4 Aim 8

5 Objectives 8

6 Course Structure 9

7 Objectives and Outcomes 11

7.1 Table of Objectives and Outcomes 11

7.2 Key Competencies 13

8 Content: Software Design and Development Stage 6 Preliminary Course 14

8.1 Concepts and Issues in the Design and Development of Software 14

8.2 Introduction to Software Development 20

8.3 Developing Software Solutions 29

9 Content: Software Design and Development Stage 6 HSC Course 31

9.1 Development and Impact of Software Solutions 31

9.2 Software Development Cycle 35

9.3 Developing a Solution Package 49

9.4 Options 52

10 Course Requirements 56

11 Post-school Opportunities 57

12 Assessment and Reporting 58

12.1 Requirements and Advice 58

12.2 Internal Assessment 59

12.3 External Examination 59

12.4 Board Requirements for the Internal Assessment Mark in Board Developed Courses 60

12.5 Assessment Components, Weightings and Tasks 61

12.6 HSC External Examination Specifications 62

12.7 Summary of Internal and External Assessment 63

12.8 Reporting Student Performance Against Standards 64

13 Glossary 65

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. Software development is a distinctive field within the Computing discipline. Stage 6 students who wish to move into this field are at an advantage if they understand the field.

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.

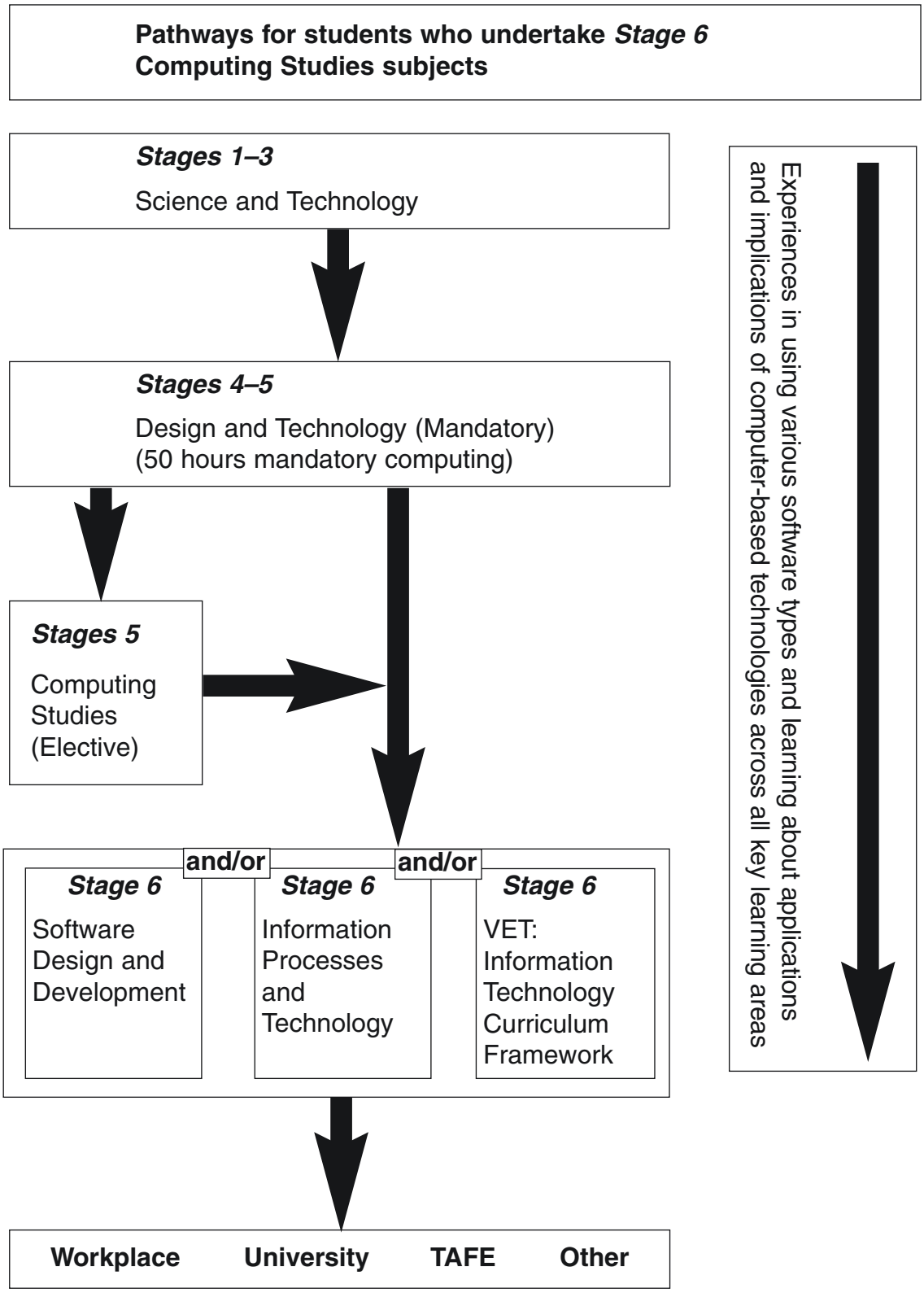
Computing is an area of rapid growth and change. While a variety of computer applications are used in this subject, they are not the primary focus. The focus of this subject is the development of computer-based solutions that require the design of computer software.

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 useful.

The subject is intended for both genders. 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.

Software Design and Development promotes intellectual, social and ethical growth in students. The subject has been developed from an area of identified student interest. It provides them with 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. Study of this subject will enable students to take part in debates on software development in society. 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

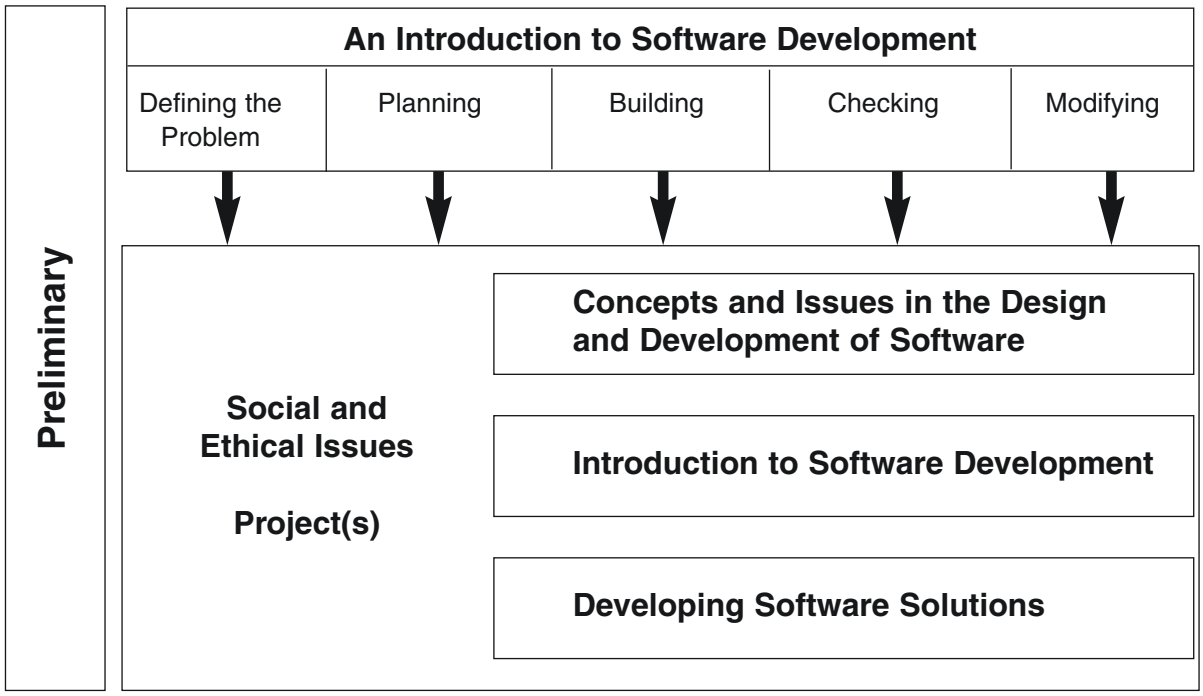
Students will develop:

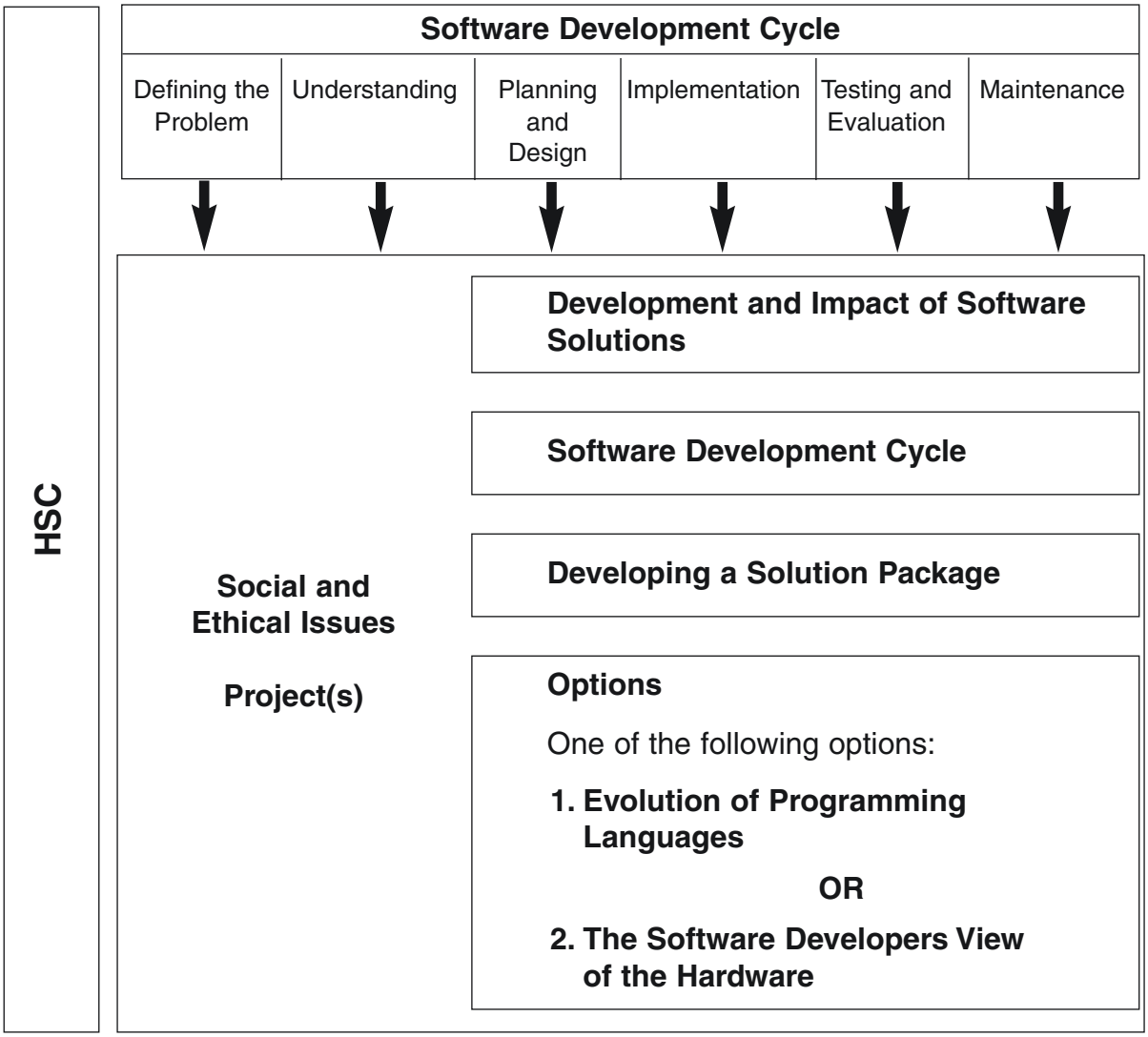
1. knowledge and understanding about how software solutions utilise and interact with other elements of computer systems
2. knowledge and understanding of the historical developments that have led to current practices in software design and development, and of emerging trends and technologies in this 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	**Development and Impact of Software Solutions 15%** • Social and ethical issues • Application of software development approaches
Introduction to Software Development 50% • Defining the problem and planning software solutions • Building software solutions • Checking software solutions • Modifying software solutions	**Software Development Cycle 40%** • Defining and understanding the problem • Planning and design of software solutions • Implementation of software solutions • Testing and evaluation of software solutions • Maintenance of software solutions
Developing Software Solutions 20%	**Developing a Solution Package 25%**
	Options 20% One of the following options: 1. Evolution of programming languages OR 2. The Software Developer's view of the hardware


Preliminary
An Introduction to Software Development
Defining the Problem
Planning
Building
Checking
Modifying
Concepts and Issues in the Design and Development of Software
Social and Ethical Issues
Introduction to Software Development
Project(s)
Developing Software Solutions


HSC
Software Development Cycle
Defining the Problem
Understanding
Planning and Design
Implementation
Testing and Evaluation
Maintenance
Development and Impact of Software Solutions
Software Development Cycle
Developing a Solution Package
Social and Ethical Issues
Project(s)
Options
One of the following options:
1. Evolution of Programming Languages
OR
2. The Software Developers View of the Hardware

7 Objectives and Outcomes

7.1 Table of Objectives and Outcomes

Objectives	Preliminary outcomes	HSC outcomes
Students will develop:	A student:	A student:
1. knowledge and understanding about how software solutions utilise and interact with other elements of computer systems	P1.1 describes the functions of hardware and software	H1.1 explains the interrelationship between hardware and software
	P1.2 describes and uses appropriate data types	H1.2 differentiates between various methods used to construct software solutions
	P1.3 describes the interactions between the elements of a computer system	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	H2.1 describes the historical development of different language types
	P2.2 explains the effects of historical developments on current practices	H2.2 explains the relationship 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	H4.1 identifies needs to which software solutions are appropriate
	P4.2 investigates a structured approach in the design and implementation of a software solution	H4.2 applies appropriate development methods to solve software problems
	P4.3 uses a variety of development approaches to generate software solutions and distinguishes between these approaches	H4.3 applies a modular approach to implement well structured software solutions and evaluates their effectiveness

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 role of personnel 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 relationship between the roles of people involved in the software development cycle H6.2 communicates the processes involved in a software solution to an inexperienced user H6.3 uses a collaborative approach during the software development cycle H6.4 develops 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

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, language used in documentation should all be considered again at relevant parts in the course.

Outcomes

A student:

P2.2 explains the effects of historical developments on current practices

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

P6.1 describes the role of personnel involved in software development.

Students learn about:	Students learn to:
Ergonomics • effects of prolonged use of software, including RSI and injuries created by overuse • procedures to prevent and minimise injuries • ergonomically designed and placed equipment • ergonomic issues regarding software design: – acceptable response time in software – 'user friendly' software, including: - ease of use - appropriate messages to the user - consistency of the user interface	• identify sound ergonomic practices when using computers • assess the ergonomic needs of the user when developing software
Intellectual property • software licence agreements, including: – licence terminology	• debate the issues relating to intellectual property

Students learn about:	Students learn to:
– legal aspects – use of software covered by a licence agreement • origin of software design ideas – evolution of existing concepts, including: - GUI interface - search engines – new and exciting approaches, including: - visicalc - web browsers - presentation software • 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 conditions that apply, including: – the Internet – books and magazines – shareware	• use software in an ethically and legally correct manner
Inclusivity • the need for software design and development to be inclusive – cultural perspectives – economic perspectives – social perspectives – gender perspectives – disability perspectives • the general strengths brought to the field of software design and development, including: – communication skills – ability to work in teams – creativity – design skills – problem-solving skills – attention to detail	• evaluate existing software interfaces in terms of its inclusivity

8.1.2 Hardware and software

This topic is intended to introduce two of the components of a computer system, hardware and software. Hardware and software are two different but dependent components of a computer system — they cannot be used in isolation. This topic looks at the different parts of a computer system and their relationship to software design.

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 explains the effects of historical developments on current practices

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

P6.1 describes the role of personnel involved in software development.

Students learn about:	Students learn to:
Hardware • the function of hardware within a computer system, namely: – input – output – process – storage – control • the operation of a variety of input devices, output devices, storage devices and CPU components • the current trends and developments in computer hardware	• describe how data is captured, stored and manipulated on a variety of hardware devices • competently use computer hardware, selecting appropriate hardware for specific tasks
Software • system software, including utility software • applications packages and custom-designed software • generations of programming languages, namely: – machine – assembler – higher level languages – declarative languages	• competently use a range of software • describe the development of subsequent generations of programming languages

Students learn about:	Students learn to:
• event driven versus sequential approach • the need for translation – compilation – interpretation – incremental compilation • characteristics of different operating systems, including: – command-based or graphical user interface – multi-tasking • current trends in the development of software and operating systems	• appraise the effect of the operating system on the tasks that the system can perform • interpret and use an ASCII table
The relationship between hardware and software • processing of software instructions by hardware – the 'fetch-execute' cycle • the initiation and running of an application – start fetch-execute cycle – locate on disk – load into RAM – display the start screen – wait for user input • the existence of minimum hardware requirements to run some software • elements of a computer system, including: – hardware – software – data – procedures – personnel and their role in software design and development	• identify the elements of a computer system • describe the significance of each element in the software solution using a case study approach

8.1.3 Software development approaches

There are a number of different approaches that can be taken when developing software. Four are prescribed for study in this course. The approach used for a given software solution will reflect the level of ability of those developing the software, its purpose and its users. There are many ways in which software is commercially developed, from an ad-hoc approach to the very formalised structured approach. This topic introduces students to some of the alternative approaches and the relevance of each.

Outcomes

A student:

P2.2 explains the effects of historical 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 role of personnel involved in software development.

Students learn about:	Students learn to:
The structured approach to software solutions • program development cycle for the structured approach, including defining the problem, planning, building, checking and modifying • characteristics of the structured approach, including: – long time periods – large-scale projects – large budgets • involvement of personnel, including analysts, designers, programmers, users and management • team approach **The prototyping approach to software solutions** • characteristics of the prototyping approach, including: – non-formal – shorter time period – small-scale projects – small budgets	• identify each of these stages in practical programming exercises

Students learn about:	Students learn to:
• involvement of personnel, including programmer and users • links with structured approach	• design and develop a limited prototype as a demonstration of a solution to a specified problem
The rapid applications software development approach • characteristics of the rapid approach, including: – lack of formal stages – coding languages used – relationship of programmer to end user – short time period – small-scale projects – low budgets • involvement of personnel, including developer and end user	• use an existing software package to develop a customised solution
End user approach to software development • characteristics of the end user approach, including: – use of standard software packages – lack of formal stages – short time period – potential long-term, small-scale project – low budgets – end user is the developer	• select appropriate software development approaches for specific purposes

8.2 Introduction to Software Development

All software development approaches include the phases of defining the problem, planning, building, checking and modifying. There are variations in the time, sequence and organisation of these phases in each of the four 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 modelling and simulation, hypermedia tools, publishing on the World Wide Web and customisation of application packages through scripting or writing modules.

8.2.1 Defining the problem and planning 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 Methods of Algorithm Description (see page 56). 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 explains the effects of historical 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:
Defining the problem • understanding the problem • identification of inputs and required outputs • determining the steps that, when carried out, will solve the problem	• determine the inputs and outputs required for a particular problem

Students learn about:	Students learn to:
Abstraction/Refinement • the top-down approach to solution development • refinement of a proposed solution • modification of an existing solution	• develop a systematic approach to the development of a solution
Data types • data types used in solutions, including: – integer – string – floating point – boolean – date and currency format • data structures, including: – one-dimensional array – record – sequential files • limits of particular data types • integer representation in binary, decimal, octal and hexadecimal • the impact of hardware/software limits on different data types	• select the most appropriate data type for the solution to a particular problem and discuss the merit of the chosen type
Structured algorithms • methods for representing algorithms: – pseudocode – flowcharts	• interpret and create algorithms represented in both pseudocode and flowcharts
• control structures: – sequence – selection (binary, multiway) – iteration (pre-test, post-test), including: for … next loops • software structure – subroutines – modularity	• identify control structures in an algorithm
• use of standard algorithms, including: – load and print an array – process records from a sequential file	• detect logic errors in an algorithm by performing a desk check
• checking the algorithm for errors • historical events that led to the development of a structured approach to algorithm design	• gather solutions from a number of sources and modify them to form an appropriate solution to a specified problem

8.2.2 Building software solutions

The building phase could involve a range of activities from modifying existing code to the development of new code. In order to build 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 design concepts being taught, not one that is currently fashionable. In some cases, this may be a scripting language for an applications package. Language choice will also be affected by the type of translation to be used, and whether or not a sequential or an event-driven approach is to be used. 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 meet the course requirements as specified in Software Specifications (see page 56). For every algorithm that is implemented, the specified user interface will need to be developed along with documentation that explains what has taken place during the building phase. Relevant social and ethical issues should be revisited, particularly with reference to appropriate interface design, language used in the interfaces and issues related to using others' designs and software

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

P5.1 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 role of personnel 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: – BNF – EBNF – Railroad diagrams	• use meta-language statements from manuals and help files to develop syntactically correct code
• reading and writing statements in meta-languages	
• the syntax of the statements used to represent the control structures, including: – sequence – selection (binary, multiway) – iteration (pre-test, post-test) – combinations of these	• verify the syntax of a command using meta-language statements
• the syntax of the statements used to define and use a range of data types, including: – integer – string – floating point – one-dimensional array – record – sequential files	• generate appropriate source code by: – using a programming environment to generate and execute code – coding an algorithm into the chosen programming language – using different data types in solutions
Error correction techniques	
• types of coding errors, including: – syntax errors – runtime errors – logic errors	
• stubs – used to check the connection between modules of code	• trace the output of a given code fragment and modify it appropriately • run, correct and extend existing code
• flags – used to check if a section of code has been processed – can be used as part of the logic of a solution or as a systematic error correction process	• systematically eliminate syntax errors so that a program can be executed • test a program with boundary values to detect runtime errors • detect and correct logic errors in program code by using a systematic error correction process
• debugging output statements: – additional print statements in the code that help in telling what part of the code has been executed or for interrogating variable contents at a particular point in the program's execution	• use automated debugging features in programming environments

Students learn about:	Students learn to:
Libraries of code • reusable code – standard routines, such as data validation, date conversion and words to numbers • combining code and modules from different sources – copying and pasting into code – ways of calling modules of code – sharing/passing variables between modules **User interface development** • consult with users • the different perspectives a user and a developer have to a program • effective user interfaces – 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: justification, font type (serif vs sans serif), size, style – recognition of relevant social and ethical issues – consistency **Documentation** • types of documentation – documentation for developers – documentation for users • internal documentation – meaningful variable names (intrinsic) – readability of code (comments, white space and indentation) • online help	• develop standard routines for reuse • create solutions to problems using existing code with minimal change or additions • represent code from different sources as an algorithm, to assist in understanding its purpose • solve problems that require the creation of a user interface • evaluate the effectiveness of screens used in commercially available software • design screens incorporating good design and ergonomic features • document code for different audiences • fully document a solution that has been developed in the classroom • use application packages to document a solution • interpret code and documentation prepared by others

8.2.3 Checking software solutions

Students should check their code using test data that test all possibilities. Live testing of programs should take place so that environment problems can be identified and removed. Students should also be checking that original requirements are being met. Specifications for a problem and a solution to the problem could be given to students and they could 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 works fully 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.

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 role of personnel 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:
Test data • selecting data for which the expected output is known • the need for thorough test data • the selection of appropriate test data, including: – data that test all the pathways through the algorithm – data that test boundary conditions — upper and lower values and values upon which decisions are based – data where the required answer is known • testing both algorithms and coded solutions with test data, such as: – desk checking an algorithm – stepping through a coded solution line by line	• determine the expected result given the test data • create a set of appropriate test data and use them to verify the logic in a solution • use test data on algorithms and coded solutions

Students learn about:	Students learn to:
Evaluation of design • comparing different solutions to the same problem – different interpretations of the design specifications – the advantages and disadvantages of different approaches to reaching the solution • peer checking • structured walk through • desk checking **Evaluation of implemented solution** • checking the solution to see if it meets the original design specifications • user feedback • social and ethical perspective	• communicate solutions to others • critically evaluate their work and that of their peers and share good aspects of their solutions using elegant aspects of other students' solutions

8.2.4 Modifying 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. 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 their solutions as a means of assessing their understanding of their original solution. 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 explains the effects of historical 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 role of personnel 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 maintenance coding • 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 **Social and ethical implications** • plagiarism **Features in source code that improve its maintainability**, including: • use of variables instead of literal constants	• identify features in code, scripts or macros that allow it to be easily maintained and explain how this can be achieved • create solutions to ensure ease of maintenance

Students learn about:	Students learn to:
• use of meaningful variable names • explanation comments in the code • use of standard control structures • a clear and uncluttered mainline • one logical task per subroutine	
Interpretation • reading original documentation in order to understand the code – documents for the user (including user manuals) – documents for software developers • reading original algorithms to identify: – inputs to the algorithm – the types of variables used – processes used – outputs • creating algorithms for source code when they are not available	• modify original statements obtained from a variety of sources • convert a fragment of source code, macro or script into its equivalent algorithm • define the purpose of the code, macro or script to be maintained
Documentation • using supplied documentation to: – identify the control structures that have been used – explain how variables have been used	

8.3 Developing Software Solutions

The project(s) will build students' understanding of the content in the other topics in the course and allow for practical implementation of theory.

Working in teams is common in the computing field beyond school. In order to be a successful member of a team, students need to be able to communicate well with others and to act in a social and ethical way. Project(s) are areas in which students may be given these opportunities.

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
P6.1 describes the role of personnel 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:
Implementing projects the steps in implementing project(s) include: • defining the problem – understanding the problem – identification of inputs, processes and outputs to be applied to the problem • planning – identification of a suitable development approach – design of appropriate algorithms – determination of appropriate data structures – identification of relevant subroutines – the design of test data and expected output – the desk check of algorithms – identification of existing code that can be used	

Students learn about:	– ergonomics **Students learn to:**
• building – implementation of the solution in an appropriate language – testing of the solution using test data – documenting the solution, including algorithms, tutorial, test data and expected output, data dictionary • checking – testing of the solution using test data – evaluation of the completed solutions • modifying – changing the solution to meet the specifications **Project management techniques** • identification of tasks • identification of techniques to assist project management, including: – Gantt charts – logbooks – identification of sub-goals • allocation of resources • identification of major milestones and stumbling blocks • regular backup • response to difficulties • regular reporting • evaluation **Project documentation** • relevant documentation may include the use of: – algorithms – Gantt charts – manuals – systems documentation – data dictionaries – diaries – CASE-tools **Social and ethical issues related to project work** • relevant issues may include: – ease of use – gender bias – accessibility of technical language – copyright	• design and implement a software solution to a selected problem using project implementation steps • use Gantt charts and logbooks • devise a management plan and use it when undertaking a software development project • use appropriate application packages in creating documentation to support the development of a project • prepare suitable documentation to accompany software solutions • ensure relevant social and ethical issues have been addressed • evaluate the project in relation to the original understanding of the problem • evaluate the quality of the solution

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 computer use. In addition to acting in socially responsible and ethical ways, students should implement these values into their broader use of computers. Students should be able to identify relevant social and ethical issues and participate in current debates. 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. It is intended that all of these issues be continually revisited within each topic in the HSC course.

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:
Rights and responsibilities of software developers • authorship • reliability • quality • response to problems • code of conduct • viruses	• identify the impact on consumers of inappropriately developed software
Software piracy and copyright • concepts associated with piracy and copyright, including: – intellectual property – plagiarism – shareware – public domain – ownership versus licensing – copyright laws – reverse/backwards engineering – decompilation – licence conditions – network use	• interpret copyright agreements and develop personal practices that reflect current laws
• various national perspectives to software piracy and copyright laws • the relationship between copyright laws and software license agreements	• acknowledge all sources in recognition of the intellectual contribution of authors
The software market • maintaining market position • the effect on the marketplace	
Significant social and ethical issues • national and international legal action resulting from software development • public issues, including: – the year 2000 problem – computer viruses – reliance on software	• debate current issues relevant to software development

9.1.2 Application of software development approaches

Students should be aware of the advantages and disadvantages of each of the different software development approaches introduced in the Preliminary course. Students will complete a case study of software being developed by a team of people. Particular emphasis should be placed on the people involved, how they interact and the skills they possess. Current trends in software development will also be considered.

Outcomes

A student:

H1.2 differentiates between various methods used to construct software solutions

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

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 relationship between the roles of people involved 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: – the structured approach – prototyping – rapid applications development – end user development – combinations of any of the above • methods of implementation – direct cut over – parallel – phased – pilot • current trends in software development, for example: – outsourcing – popular approaches – popular languages – employment trends – networked software – customised off-the-shelf packages • use of CASE tools and their application in large systems development – software versions – data dictionary – test data – production of documentation	• compare and determine the most appropriate software development approach for a given scenario • communicate their understanding of a commercial system studied using a case study approach by: – describing how the skills of the various personnel contribute to the overall development of a computer-based system – critically evaluating the effectiveness of the response to the social and ethical issues raised by this system • make informed comment on current trends in software development

9.2 Software Development Cycle

While many of the students who will study this course may have had some previous experience in the development of software, few will have done so using the formal methods that make up the software development cycle. This approach to software development will empower students to undertake much more complex development projects, knowing that the developed system will be in a standard maintainable format. Students should draw on the skills of others to assist them in this process. The topics that come together to form this cycle are the fundamentals of the HSC course. These topics should not be studied in isolation or in a sequential fashion. Students should be exposed to the content in a cyclic fashion. The project requires that students follow and implement the cycle from beginning to end. Areas for investigation here could include modelling and simulation, the production of games, hypermedia tools, publishing on the World Wide Web and customisation of application packages through scripting or writing modules.

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 also 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 Software Specifications (see page 56) and should produce documentation able to be interpreted by developers, maintainers and users as required. 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 relationship between the roles of people involved in the software development cycle

H6.2 communicates the processes involved in a software solution to an inexperienced user

H6.3 uses a collaborative approach during the software development cycle

H6.4 develops effective user interfaces, in consultation with appropriate people.

Students learn about:	Students learn to:
Defining the problem • identifying the problem – needs – objectives – boundaries • determining the feasibility of the solution – is it worth solving? – constraints – budgetary – operational – technical – scheduling – possible alternatives – social and ethical considerations	• develop and interpret design specifications from a user's perspective, considering: – screen design – appropriate messages – appropriate icons – relevant data formats for display – ergonomic issues – relevance to the user's environment and computer configuration – social and ethical issues
Design specifications • the developer's perspective in consideration of: – data types – algorithms – variables • the user's perspective	• evaluate the extent to which a proposed system will meet user needs
Modelling • representing a system using diagrams, including: – Input Process Output (IPO) diagrams – story boards – data flow diagrams – systems flowcharts – screen designs – consideration of use of a limited prototype	• differentiate between the different forms of systems documentation and the purposes for which each is intended • interpret a system presented in a diagrammatic form • create a diagrammatic representation for a system using an appropriate method
Communication issues, including: • the need to empower the user • the need to acknowledge the user's perspective • enabling and accepting feedback	• effectively communicate with users regarding a proposed software solution

9.2.2 Planning and design of software solutions

To solve complex problems, students need to develop a strategy. They need to be able to identify inputs and outputs, to select and describe 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 that will be used in eventually implementing the algorithm. A well-developed algorithm can be implemented in any number of languages, while transferring code from one language to another is a more difficult process. 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.

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 a collaborative approach during the software development cycle.

Students learn about:	Students learn to:
Standard algorithms for searching and sorting • standard logic used in software solutions, namely: – finding maximum and minimum values in arrays – processing strings (extracting, inserting, deleting) – file processing, including sentinel value – linear search – binary search – bubble sort – insertion sort – selection sort	• recognise the logic in a standard approach (such as a sort or search)
Custom-designed logic used in software solutions • requirements to generate these include: – identification of inputs, processes and outputs – representation as an algorithm – definition of required data structures – use of data structures, including multi-dimensional arrays, arrays of records, files (sequential and relative/random) – use of random numbers – thorough testing	• apply standard approaches as part of the solution to complex problems • document the logic required to solve problems, including: – file handling and management – random number generators – multi-dimensional arrays – nesting of control structures • develop a suitable set of test data and desk check algorithms that include complex logic • select an appropriate data structure to solve a given problem • develop a standard module and document its use • correctly incorporate a standard module into a more complex solution, passing parameters effectively
Standard modules (library routines) used in software solutions • requirements for generating or subsequent use include: – identification of appropriate modules – consideration of local and global variables – appropriate use of parameters (arguments) – appropriate testing using drivers – thorough documentation	

Students learn about:	Students learn to:
Customisation of existing software solutions • identification of relevant products • customisation • cost effectiveness	• evaluate the effectiveness of using commercially developed software
Documentation of the overall software solution • tools for representing a complex software solution include: – algorithm descriptions – system flowcharts – structure diagrams – data flow diagrams – data dictionary	• represent a software solution in diagrammatic form • identify the parts of the system that require software to be custom designed and developed • select and use appropriate CASE software to assist in the development of a software solution
Selection of language to be used • event-driven software – driven by the user – program logic • sequential approach – defined by the programmer • relevant language features • hardware ramifications • Graphical User Interface (GUI)	

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. The translation method being used should be recognised, particularly in the case of code. 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

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

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

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 a collaborative approach during the software development cycle.

Students learn about:	Students learn to:
Interface design in software solutions • the design of individual screens, including: – identification of data required – current popular approaches – design of help screens – audience identification – consistency in approach	• select either a sequential or event-driven approach and an appropriate language to effectively solve the problem • design and evaluate effective screens for software solutions
Language syntax required for software solutions • use of BNF, EBNF and railroad diagrams to describe the syntax of new statements in the chosen language • commands incorporating the definition and use of: – multi-dimensional arrays – arrays of records – files (sequential and relative/random) – random number generators	• utilise the correct syntax for new commands using the metalanguage specification • produce syntactically correct statements • implement a solution utilising a complex algorithm
The role of the CPU in the operation of software • machine code and CPU operation – instruction format – use of registers and accumulators – use of program counter and fetch-execute cycle – addresses of called routines – linking, including use of DLL's	• recognise and interpret machine code instructions
Translation methods in software solutions • different methods include: – compilation – incremental compilation – interpretation • the translation process • advantages and disadvantages of each method	• choose the most appropriate translation method for a given situation • utilise the features of both a compiler and an interpreter in the implementation of a software solution

Students learn about:	Students learn to:
Program development techniques in software solutions • structured approach to a complex solution, including: – one logical task per subroutine – stubs – flags – isolation of errors – debugging output statements – elegance of solution – writing for subsequent maintenance • the process of detecting and correcting errors, including: – syntax errors – logic errors – peer checking – desk checking – use of expected output – run time errors, including: - arithmetic overflow - division by zero - accessing inappropriate memory locations • the use of software debugging tools, including: – use of breakpoints – resetting variable contents – program traces – single line stepping	• justify the use of a clear modular structure with separate routines to ease the design and debugging process • use drivers to test specific modules, before the rest of the code is developed • differentiate between the different types of errors encountered during the testing phase • 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
Documentation of a software solution • forms of documentation, including: – process diary – user documentation – self-documentation of the code – technical documentation, including source code, algorithms, data dictionary and systems documentation – documentation for subsequent maintenance of the code • use of application software to assist in the documentation process – use of CASE tools	• produce user documentation (utilising screen dumps) that includes: – a user manual (topics presented in order of difficulty) – a reference manual (all commands in alphabetic order) – an installation guide – a tutorial to introduce new users to the software • identify the personnel who would be likely to use the different types of documentation

Students learn about:	Students learn to:
Hardware environment to enable implementation of the software solution • hardware requirements – minimum configuration – possible additional hardware – appropriate drivers or extensions	• recognise the need for additional hardware
Emerging technologies • hardware • software • their effect on: – human environment – development process	• assess the effect of an emerging technology on society

9.2.4 Testing and evaluation 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 environment problems can be identified and removed. Students should also be checking that original requirements are being met. 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 relationship between the roles of people involved in the software development cycle

H6.2 communicates the processes involved in a software solution to an inexperienced user

H6.3 uses a collaborative approach during the software development cycle

H6.4 develops effective user interfaces, in consultation with appropriate people.

Students learn about:	Students learn to:
Testing the software solution • comparison of the solution with the original design specifications • generating relevant test data for complex solutions • levels of testing – unit or module – program – system • the use of live test data to test the complete solution: – larger file sizes – mix of transaction types – response times – volume data – interfaces between modules – comparison with program test data • benchmarking • quality assurance	• differentiate between systems and program test data • test their solution with the test data created at the design stage, comparing actual output with that expected
Reporting on the testing process • documentation of the test data and output produced – 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 users, facilitating open discussion and evaluation

9.2.5 Maintenance of software solutions

Modifications to code, macros and scripts are often required. Often these are not made by the original developers. Under these circumstances, original documentation is of importance, as is the structure and self-documentation of the commands to be updated. Students should be given opportunities to modify their own code, macros and scripts and experience modifying the code, macros and scripts of others, supported by varying degrees of documentation.

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 relationship between the roles of people involved in the software development cycle

H6.2 communicates the processes involved in a software solution to an inexperienced user

H6.3 uses a collaborative approach during the software development cycle

H6.4 develops effective user interfaces, in consultation with appropriate people.

Students learn about:	Students learn to:
Modification of code to meet changed requirements • identification of the reasons for change in code, macros and scripts • location of section to be altered • determining changes to be made • implementing and testing solution	• read and interpret others' code, macros and scripts • design, implement and test modifications • recognise the cyclical approach to maintenance
Documentation of changes • source code, macro and script documentation • modification of associated hard copy documentation and online help • use of CASE tools to monitor changes and versions	• document modifications with dates and reasons for change

9.3 Developing a Solution Package

The project(s) in the HSC course is intended to reinforce the content covered in the other topics in the course. Students need to experience working as part of a team, 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 and to act in a social and ethical way. The project is one area in which students may be given these opportunities. The project(s) will build students' understanding of the content dealt with in the other topics in the course and should be undertaken 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 relationship between the roles of people involved in the software development cycle

H6.2 communicates the processes involved in a software solution to an inexperienced user

H6.3 uses a collaborative approach during the software development cycle

H6.4 develops 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 the problem and its solution, including: • defining the problem – identification of the problem – idea generation – communication with others involved in the proposed system • understanding – interface design – communication with others involved in the proposed system – representing the system using diagrams – selection of appropriate data structures – applying project management techniques – consideration of all social and ethical issues • planning and design – interface design – selection of software environment – identification of appropriate hardware – selection of appropriate data structures – production of data dictionary – definition of required validation processes – definition of files — record layout and creation – algorithm design – inclusion of standard or common routines – use of software to document design – identification of appropriate test data – enabling and incorporating feedback from users at regular intervals – consideration of all social and ethical issues – applying project management techniques	• define the problem and investigate alternative approaches to a software solution • select an appropriate solution • produce an initial Gantt chart • use a logbook to document the progress of their project • document the software solution • generate a fully documented design for their project after communication with other potential users

Students learn about:	Students learn to:
Systems implementation Implementing the software solution by: • implementation – production and maintenance of data dictionary – inclusion of standard or common routines – use of software to document design – translating the solution into code – creating online help – program testing – reporting on the status of the system at regular intervals – applying project management techniques – enabling and incorporating feedback from users at regular intervals – completing all user documentation for the project – consideration of all social and ethical issues – completing full program and systems testing • maintenance – modifying the project to ensure an improved solution	• 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 • communicate effectively with potential users at all stages of the project to ensure that it meets their requirements • ensure that relevant ethical and social issues are addressed appropriately

9.4 Options

The option topic in this course extends students' software development experiences in one of two dimensions. Students selecting the Evolution of Programming Languages option will broaden their understanding of the different types of programming languages by looking at different approaches to programming languages and the reasons for their development. Option 2 — The Software Developer's View of the Hardware — extends students' understanding of the layers 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 — Evolution of Programming Languages

This topic offers students the opportunity to look at approaches utilised by the 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 approach to solve a specific problem.

Outcomes

A student:

H1.2 differentiates between various methods used to construct software solutions

H2.1 describes the historical developments of different language types

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

H4.1 identifies needs to which software solutions are appropriate

H4.2 applies appropriate development methods to solve software problems

Students learn about:	Students learn to:
Historical reasons for the development of the different paradigms • a need for greater productivity • recognition of repetitive standard programming tasks • a desire to solve different types of problems (eg AI) • the recognition of a range of different basic building blocks • emerging technologies **Basic building blocks** • variables and control structures (imperative) • functions (functional) • facts and rules (logic) • objects, with data and methods or operations (object oriented) **Effect on programmers' productivity** • speed of code generation • approach to testing • effect on maintenance • efficiency of solution once coded • learning curve (training required) **Paradigm specific concepts** • logic paradigm – (eg Prolog, expert system shells) – heuristics – goal – inference engine – backward/forward chaining • object oriented programming – (eg C++, Delphi, Java) – methods – classes – inheritance – polymorphism – encapsulation – abstraction • functional (eg LISP, APL) – functions	• recognise representative fragments of code written in a particular paradigm • differentiate between the different paradigms • evaluate the effectiveness of each paradigm in meeting its perceived need • identify an appropriate paradigm relevant for a given situation • interpret a fragment of code, and identify and correct logic errors • modify fragments of code written using an example of a particular paradigm to reflect changed requirements • for current and emerging languages, identify an appropriate paradigm

9.4.2 Option 2 — The Software Developer's View of the Hardware

This topic looks in much more depth at how the hardware is utilised by the software instructions to achieve the desired outcomes. In the section, Implementation of Software Solutions, students are introduced to how the CPU processes instructions. This topic allows students to investigate further how the basic arithmetic processes and storage of data is performed by electronic circuitry. Students should recognise that the design of such circuitry follows the same cyclic process as that of 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.

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.

Students learn about:	Students learn to:
Representation of data within the computer • character representation, namely: – ASCII – hexadecimal • integer representation, including: – sign and modulus – one's complement – two's complement • representation of fractions, namely: – floating point or real • binary arithmetic, including: – addition – subtraction using two's complement representation – multiplication, shift and add – division, shift and subtract	• convert integers between binary and decimal representation • interpret the binary representation of data • recognise situations in which data can be misinterpreted by the software • perform arithmetic operations in binary

Students learn about:	Students learn to:
Electronic circuits to perform standard software operations • logic gates, including: – AND, OR, NOT, NAND, NOR, XOR • truth tables • 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 as a memory store	• generate truth tables for a given circuit • describe the purpose of a circuit from its truth table • design a circuit to solve a given problem and use a truth table to verify the design • explain how a flip-flop can be used in the storage and shifting of a bit in memory • build and test a circuit using integrated circuits or use a software package • simulate the testing of a circuit for both user-designed circuits and the specialty circuits • recognise the cyclical approach to circuit design • modify an existing circuit design to reflect changed requirements
Programming of hardware devices • the input data stream from sensor and other devices – header information – data characters – trailer information – control characters – hardware specifications – documentation • processing of data stream – the need to recognise and strip control characters – counting the data characters – extracting the data • generating output to an appropriate output device – required header information – required control characters – data – required trailer information • control systems – responding to sensor information – specifying motor operations • printer operation – control characters for features, including page throw, font change, line spacing • specialist devices with digital input and/or output	• interpret a data stream for a device for which specifications are provided • generate a data stream to specify particular operations for a hardware device, for which specifications are provided • modify a stream of data to meet changed requirements, given the hardware specifications • cause a hardware device to respond in a specified fashion

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 on practical activities using the computer, 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 on the Board of Studies website (www.boardofstudies.nsw.edu.au) following initial publication in an edition of the Board Bulletin.

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

12.1 Requirements and Advice

The information in this section of the syllabus relates to the Board of Studies' requirements for assessing and reporting achievement in the Preliminary and HSC courses for the Higher School Certificate.

Assessment is the process of gathering information and making judgements about student achievement for a variety of purposes.

In the Preliminary and HSC courses, those purposes include:

- assisting student learning
- evaluating and improving teaching and learning programs
- providing evidence of satisfactory achievement and completion in the Preliminary course
- providing the Higher School Certificate results.

Reporting refers to the Higher School Certificate documents received by students that are used by the Board to report both the internal and external measures of achievement.

NSW Higher School Certificate results will be based on:

- **an assessment mark** submitted by the school and produced in accordance with the Board's requirements for the internal assessment program
- **an examination mark** derived from the HSC external examinations.

Results will be reported using a course report containing a performance scale with bands describing standards of achievement in the course.

The use of both internal assessment and external examinations of student achievement allows measures and observations to be made at several points and in different ways throughout the HSC course. Taken together, the external examinations and internal assessment marks provide a valid and reliable assessment of the achievement of the knowledge, understanding and skills described for each course.

Standards Referencing and the HSC Examination

The Board of Studies will adopt a standards-referenced approach to assessing and reporting student achievement in the Higher School Certificate examination.

The standards in the HSC are:

- the knowledge, skills and understanding expected to be learned by students — the syllabus standards
- the levels of achievement of the knowledge, skills and understanding — the performance standards.

Both syllabus standards and performance standards are based on the aims, objectives, outcomes and content of a course. Together they specify what is to be learned and how well it is to be achieved.

Teacher understanding of standards comes from the set of aims, objectives, outcomes and content in each syllabus together with:

- the performance descriptions that summarise the different levels of performance of the course outcomes
- HSC examination papers and marking guidelines
- samples of students' achievement on assessment and examination tasks.

12.2 Internal Assessment

The internal assessment mark submitted by the school will provide a summation of each student's achievements measured at points throughout the course. It should reflect the rank order of students and relative differences between students' achievements.

Internal assessment provides a measure of a student's achievement based on a wider range of syllabus content and outcomes than may be covered by the external examination alone.

The assessment components, weightings and task requirements to be applied to internal assessment are identified on page 61. They ensure a common focus for internal assessment in the course across schools, while allowing for flexibility in the design of tasks. A variety of tasks should be used to give students the opportunity to demonstrate outcomes in different ways and to improve the validity and reliability of the assessment.

12.3 External Examination

In Software Design and Development Stage 6, the external examination includes written papers for external marking. The specifications for the examination in Software Design and Development Stage 6 are on page 62.

The external examination provides a measure of student achievement in a range of syllabus outcomes that can be reliably measured in an examination setting.

The external examination and its marking and reporting will relate to syllabus standards by:

- providing clear links to syllabus outcomes
- enabling students to demonstrate the levels of achievement outlined in the course performance scale
- applying marking guidelines based on established criteria.

12.4 Board Requirements for the Internal Assessment Mark in Board Developed Courses

For each course, the Board requires schools to submit an assessment mark for each candidate.

The collection of information for the HSC internal assessment mark must not begin prior to the completion of the Preliminary course.

The Board requires that the assessment tasks used to determine the internal assessment mark must comply with the components, weightings and types of tasks specified in the table on page 61.

Schools are required to develop an internal assessment program that:

- specifies the various assessment tasks and the weightings allocated to each task
- provides a schedule of the tasks designed for the whole course.

The school must also develop and implement procedures to:

- inform students in writing of the assessment requirements for each course before the commencement of the HSC course
- ensure that students are given adequate written notice of the nature and timing of assessment tasks
- provide meaningful feedback on each student's performance in all assessment tasks
- maintain records of marks awarded to each student for all assessment tasks
- address issues relating to illness, misadventure and malpractice in assessment tasks
- address issues relating to late submission and non-completion of assessment tasks
- advise students in writing if they are not meeting the assessment requirements in a course and indicate what is necessary to enable the students to satisfy the requirements
- inform students about their entitlements to school reviews and appeals to the Board
- conduct school reviews of assessments when requested by students
- ensure that students are aware that they can collect their Rank Order Advice at the end of the external examinations at their school.

12.5 Assessment Components, Weightings and Tasks

Assessment should include a range of tasks.

Preliminary Course

The suggested components, weightings and tasks for the Preliminary course are set out below.

Assessment Components	Weighting	Tasks
• knowledge and understanding about hardware and software, software development approaches, software development processes, social and ethical issues	30%	Tasks may include: • developing software solutions • interpreting and creating algorithms • maintaining student logbooks • research assignments • debates • case studies • unit tests
• design and development of software solutions	35%	
• project management techniques, including documentation, teamwork and communication	15%	
• project(s)	20%	

HSC Course

The internal assessment mark for Software Design and Development Stage 6 is to be based on the HSC course only. Final assessment should be based on a range and balance of assessment tasks.

Assessment Components	Weighting	Tasks
• knowledge and understanding about development and impact of software solutions and the software development cycle	20%	Tasks may include: • developing and modifying software solutions • interpreting and creating algorithms • maintaining student logbooks • research assignments • debates • oral presentation • case studies • industry reports • unit tests
• design and development of software solutions	35%	
• project management techniques, including documentation, teamwork and communication	20%	
• project(s)	25%	

While the allocation of weightings to the various tasks set for the HSC course is left to individual schools, the percentages allocated to each assessment component must be maintained. One task may be used to assess several components. It is suggested that 3–5 tasks are sufficient to assess the HSC course outcomes.

12.6 HSC External Examination Specifications

No calculators may be used in the examination. Flowchart templates may be used.

Software Design and Development

Time allowed: 3 hours (plus 5 minutes reading time)

The paper is divided into THREE sections

Section I (20 marks)

- There will be TWENTY multiple-choice questions.
- All questions are compulsory.
- Questions will be based on all the topics: Development and Impact of Software Solutions, Software Development Cycle, Developing a Solution Package.

Section II (60 marks)

- There will be THREE questions.
- All questions are compulsory.
- All questions are of equal value.
- Questions will be based on all of the topics: Development and Impact of Software Solutions, Software Development Cycle, Developing a Solution Package.
- All questions will consist of a number of parts requiring short structured responses.

Section III (20 marks)

- There will be TWO questions, one question on each of the options: Evolution of Programming Languages and Software Developer's View of the Hardware.
- Candidates must attempt ONE question only.
- Both questions are of equal value.
- Both questions will consist of a number of parts requiring short structured responses.

12.7 Summary of Internal and External Assessment

Internal Assessment	Weighting	External Assessment	Weighting
• knowledge and understanding about development and impact of software solutions and the software development cycle • design and development of software solutions • project management techniques, including documentation, teamwork and communication • project(s)	20 35 20 25	**Section I** 20 multiple-choice questions relating to: – Development and Impact of Software Solutions – Software Development Cycle – Developing a Solution Package **Section II** Three short structured response questions relating to: – Development and Impact of Software Solutions – Software Development Cycle – Developing a Solution Package **Section III** Students select one question from either: – Evolution of Programming Languages OR – Software Developer's View of the Hardware	20 60 20
Marks	100	Marks	100

12.8 Reporting Student Performance Against Standards

Student performance in an HSC course will be reported against standards on a course report. The course report includes a performance scale for the course describing levels (bands) of achievement, an HSC examination mark and the internal assessment mark. It will also show, graphically, the statewide distribution of examination marks of all students in the course.

Each band on the performance scale (except for band 1) includes descriptions that summarise the attainments typically demonstrated in that band.

The distribution of marks will be determined by students' performance against the standards and not scaled to a predetermined pattern of marks.

13 Glossary

Syllabus specific terms. These terms are provided to assist teachers to interpret the syllabus but are in no way intended for examination purposes.

abstraction	The hiding of detail by the presentation of a more general instance. In the programming environment, an example of this is the use of a subroutine, rather than the inclusion of detailed code
backwards/ forwards chaining	The process of arriving at a conclusion from a stated set of conditions. Backwards chaining assumes that a particular solution is true and then ask questions to verify that the necessary conditions are present. Forward chaining starts from the beginning of the facts and rules and asks questions to determine which path to follow next to arrive at a conclusion
benchmarking	A method used to measure the performance of a system or application by running it under closely controlled conditions
BNF	Backus Naur Format — a metalanguage used to specify the syntax of commands in a given language
breakpoints	A method used in software debuggers to denote a point at which the program is to temporarily halt execution. The programmer can examine or change the contents of variables at this point and then resume execution if appropriate
CASE tools	Computer Aided Software Engineering — a range of software that is used to assist the developer with a variety of tasks required as part of the development process
class	The definition of the common characteristics of a group of objects, which can be used as a ‘template’ for these objects. Objects of the same class have the same basic definition for their processes and data
decompilation	The process of taking executable machine code and generating the equivalent assembler code, so that it is more easily understood by a human. This process is often necessary when the executable code needs to be modified and the programmer does not have access to the source code
driver	A specially written routine that generates appropriate test data used to test a lower level module before the higher level modules are completed
EBNF	Extended Backus Naur Format — a more sophisticated metalanguage used to specify the syntax of commands available in a given language
encapsulation	The isolation of an object from its environment, so that changes to objects can be made without affecting other parts of the system, as long as the interface to that object remains the same

end user development	A process in which an application is developed by users who have knowledge of a relevant software package and can customise it to meet their needs
heuristics	Rules of thumb that generally leads to a correct conclusion, but which may never be able to be proved
inclusivity	A recognition of equal access
incremental compilation	A translation process used with an interpreter in which commonly executed routines are translated separately into machine code and called directly as required
inference engine	The logic used by expert system software to draw conclusions from stated facts and relevant rules
metalanguage	A means of specifying the syntax of each of the valid commands in a given language
method	The specification of a particular process to be performed on or by an object
object	In an object oriented programming environment, this refers to the data structures and procedures that apply to a specific unit in the system
operation	In an object oriented programming environment, this refers to the method or process to be performed on or by an object
paradigm	A model, used in this context to refer to a type of programming language
polymorphism	The concept that allows different objects to be used or presented in different ways at run time, depending on the users' requirements at the time
quality assurance	A set of procedures used to certify that a generated product meets specified criteria with respect to quality and reliability
rapid application development	A process in which a programmer makes use of software packages to quickly build applications to meet the users' needs
reverse engineering	The process of analysing an existing system to identify its components and their interrelationships, to allow the creation of a similar system
sentinel value	A value used to signify the end of a data list, such as 'ZZZ' or 99999
structured walkthrough	An approach used with project teams, where each developer working on a project steps the other members of the team through the work they have completed so far. It is used to ensure consistency of approach and assists in ensuring the overall quality of the project as a whole