

B O A R D O F S T U D I E S
NEW SOUTH WALES

Software Design and Development

Stage 6 Syllabus

Original published version updated:

September 1999 – Board Bulletin/Official Notices Vol 8 No 7 (BOS 54/99)

June 2009 – Assessment and Reporting information updated

The Board of Studies owns the copyright on all syllabuses. Schools may reproduce this syllabus in part or in full for bona fide study or classroom purposes only. Acknowledgement of the Board of Studies copyright must be included on any reproductions. Students may copy reasonable portions of the syllabus for the purpose of research or study. Any other use of this syllabus must be referred to the Copyright Officer, Board of Studies NSW. Ph: (02) 9367 8111; fax: (02) 9279 1482.

Material on p 5 from *Securing Their Future* © NSW Government 1997.

© Board of Studies NSW 2009

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

2009407

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	48
	9.4 Options	51
10	Course Requirements	56
11	Post-school Opportunities	57
12	Assessment and Reporting	58
13	Glossary	59

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.

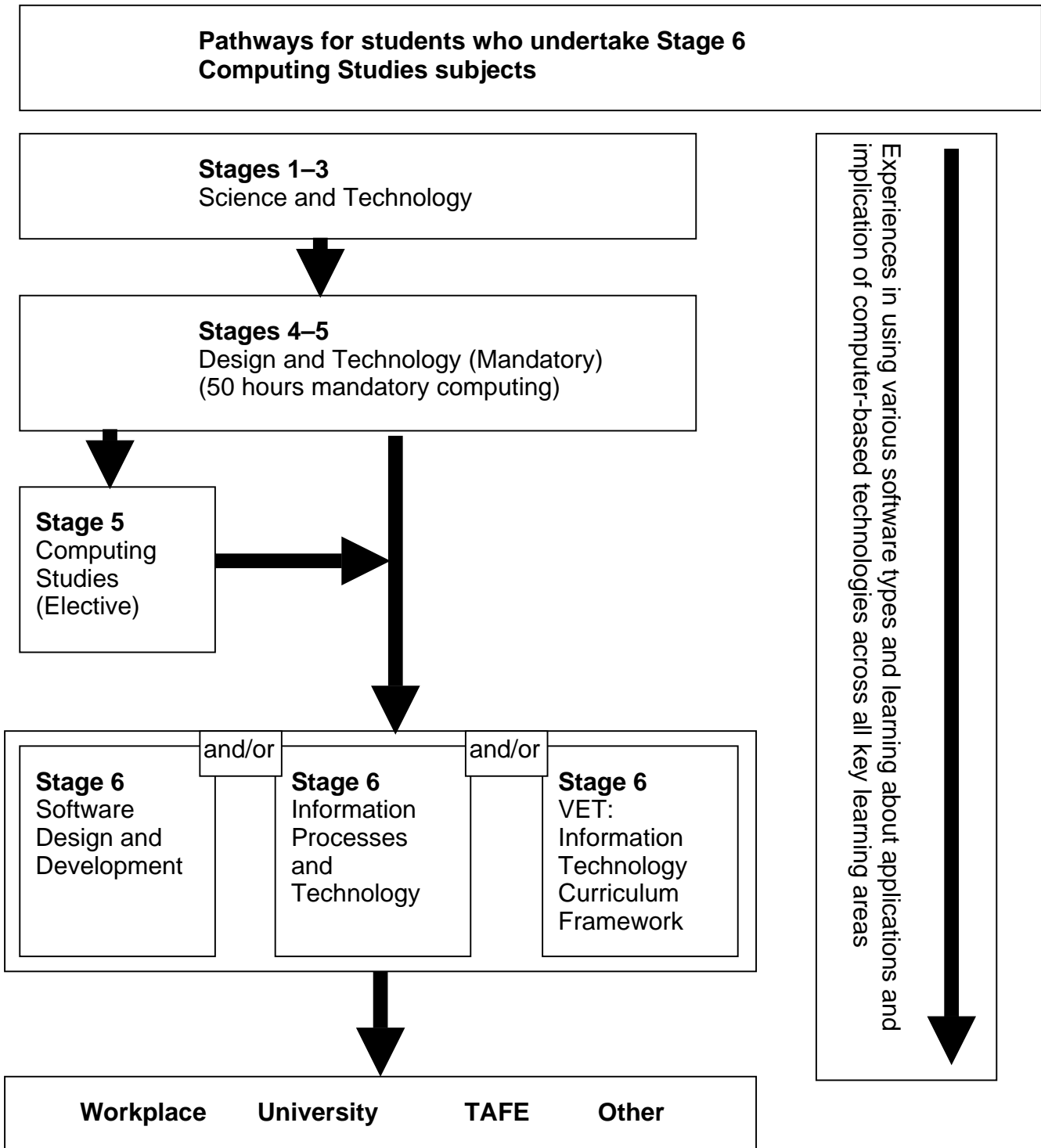
Computing is an area of rapid growth and change. While a variety of computer applications are used in this subject, they are not the primary focus. The focus of this subject is the development of computer-based solutions that require the design of computer software.

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

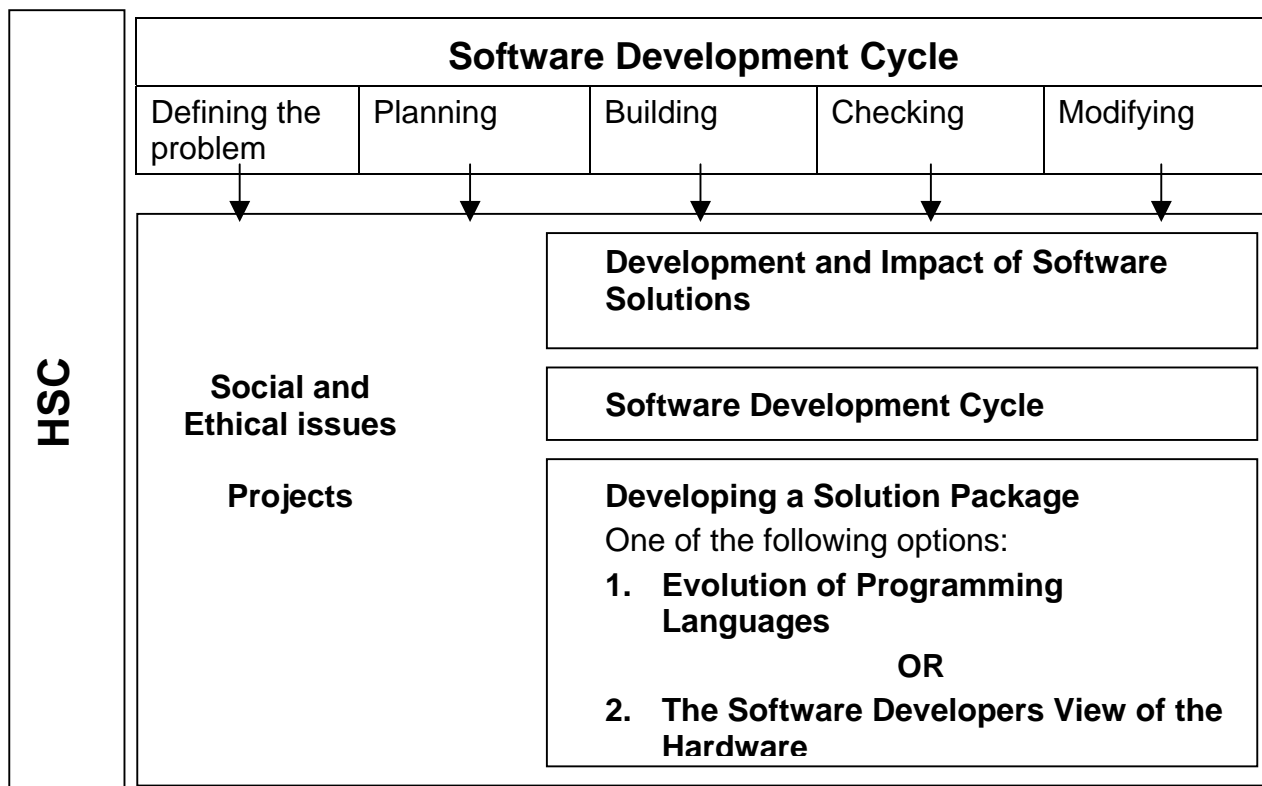
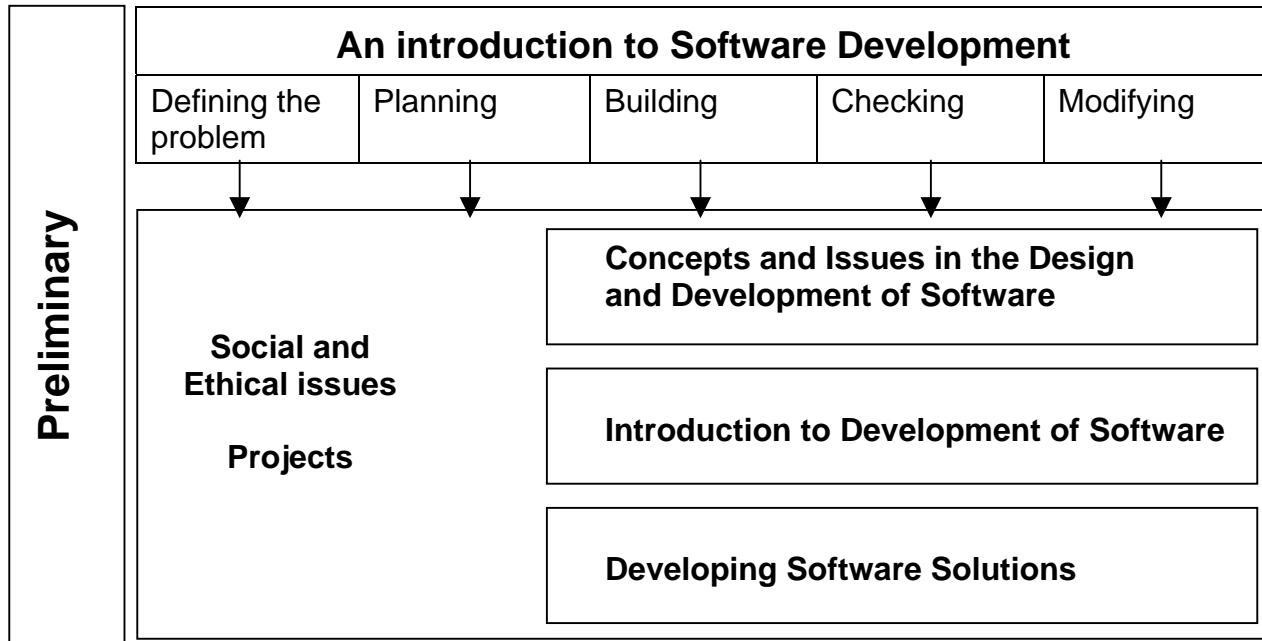
Students will develop:

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)
<p>Concepts and Issues in the Design and Development of Software 30%</p> <ul style="list-style-type: none"> • Social and ethical issues • Hardware and software • Software development approaches <p>Introduction to Software Development 50%</p> <ul style="list-style-type: none"> • Defining the problem and planning software solutions • Building software solutions • Checking software solutions • Modifying software solutions <p>Developing Software Solutions 20%</p>	<p>Development and Impact of Software Solutions 15%</p> <ul style="list-style-type: none"> • Social and ethical issues • Application of software development approaches <p>Software Development Cycle 40%</p> <ul style="list-style-type: none"> • 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 <p>Developing a Solution Package 25%</p> <p>Options 20%</p> <p>One of the following options:</p> <ol style="list-style-type: none"> 1. Evolution of programming languages <p>OR</p> <ol style="list-style-type: none"> 2. The Software Developer’s view of the hardware



7 Objectives and Outcomes

7.1 Table of Objectives and Outcomes

Objectives	Preliminary outcomes	HSC outcomes
<p>Students will develop:</p> <p>1. knowledge and understanding about how software solutions utilise and interact with other elements of computer systems</p>	<p>A student:</p> <p>P1.1 describes the functions of hardware and software</p> <p>P1.2 describes and uses appropriate data types</p> <p>P1.3 describes the interactions between the elements of a computer system</p>	<p>A student:</p> <p>H1.1 explains the interrelationship between hardware and software</p> <p>H1.2 differentiates between various methods used to construct software solutions</p> <p>H1.3 describes how the major components of a computer system store and manipulate data</p>
<p>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</p>	<p>P2.1 describes developments in the levels of programming languages</p> <p>P2.2 explains the effects of historical developments on current practices</p>	<p>H2.1 describes the historical development of different language types</p> <p>H2.2 explains the relationship between emerging technologies and software development</p>
<p>3. knowledge and understanding of legal, social and ethical issues and their effect on software design and development</p>	<p>P3.1 identifies the issues relating to the use of software solutions</p>	<p>H3.1 identifies and evaluates legal, social and ethical issues in a number of contexts</p> <p>H3.2 constructs software solutions that address legal, social and ethical issues</p>
<p>4. skills in designing and developing software solutions</p>	<p>P4.1 analyses a given problem in order to generate a computer-based solution</p> <p>P4.2 investigates a structured approach in the design and implementation of a software solution</p> <p>P4.3 uses a variety of development approaches to generate software solutions and distinguishes between these approaches</p>	<p>H4.1 identifies needs to which software solutions are appropriate</p> <p>H4.2 applies appropriate development methods to solve software problems</p> <p>H4.3 applies a modular approach to implement well structured software solutions and evaluates their effectiveness</p>

<p>5. skills in management appropriate to the design and development of software solutions</p>	<p>P5.1 uses and justifies the need for appropriate project management techniques</p> <p>P5.2 uses and develops documentation to communicate software solutions to others</p>	<p>H5.1 applies project management techniques to maximise the productivity of the software development</p> <p>H5.2 creates and justifies the need for the various types of documentation required for a software solution</p> <p>H5.3 selects and applies appropriate software to facilitate the design and development of software solutions</p>
<p>6. skills in teamwork and communication associated with the design and development of software solutions</p>	<p>P6.1 describes the role of personnel involved in software development</p> <p>P6.2 communicates with appropriate personnel throughout the software development process</p> <p>P6.3 designs and constructs software solutions with appropriate interfaces</p>	<p>H6.1 assesses the relationship between the roles of people involved in the software development cycle</p> <p>H6.2 communicates the processes involved in a software solution to an inexperienced user</p> <p>H6.3 uses a collaborative approach during the software development cycle</p> <p>H6.4 develops effective user interfaces, in consultation with appropriate people</p>

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**.

Students learn about:	Students learn to:
<ul style="list-style-type: none"> • event driven versus sequential approach • the need for translation <ul style="list-style-type: none"> – compilation – interpretation – incremental compilation • characteristics of different operating systems, including: <ul style="list-style-type: none"> – command-based or graphical user interface – multi-tasking • current trends in the development of software and operating systems <p>The relationship between hardware and software</p> <ul style="list-style-type: none"> • processing of software instructions by hardware <ul style="list-style-type: none"> – the ‘fetch-execute’ cycle • the initiation and running of an application <ul style="list-style-type: none"> – 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: <ul style="list-style-type: none"> – hardware – software – data – procedures – personnel <p>and their role in software design and development</p>	<ul style="list-style-type: none"> • appraise the effect of the operating system on the tasks that the system can perform • interpret and use an ASCII table • 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:
<p>The structured approach to software solutions</p> <ul style="list-style-type: none"> • program development cycle for the structured approach, including defining the problem, planning, building, checking and modifying • characteristics of the structured approach, including: <ul style="list-style-type: none"> – long time periods – large-scale projects – large budgets • involvement of personnel, including analysts, designers, programmers, users and management • team approach <p>The prototyping approach to software solutions</p> <ul style="list-style-type: none"> • characteristics of the prototyping approach, including: <ul style="list-style-type: none"> – non-formal – shorter time period – small-scale projects – small budgets 	<ul style="list-style-type: none"> • identify each of these stages in practical programming exercises

Students learn about:	Students learn to:
<ul style="list-style-type: none"> • involvement of personnel, including programmer and users • links with structured approach <p>The rapid applications software development approach</p> <ul style="list-style-type: none"> • characteristics of the rapid approach, including: <ul style="list-style-type: none"> – 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 <p>End user approach to software development</p> <ul style="list-style-type: none"> • characteristics of the end user approach, including: <ul style="list-style-type: none"> – 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 	<ul style="list-style-type: none"> • design and develop a limited prototype as a demonstration of a solution to a specified problem • use an existing software package to develop a customised solution • 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:
<p>Defining the problem</p> <ul style="list-style-type: none"> • understanding the problem • identification of inputs and required outputs • determining the steps that, when carried out, will solve the problem 	<ul style="list-style-type: none"> • determine the inputs and outputs required for a particular problem

Students learn about:	Students learn to:
<p>Abstraction/Refinement</p> <ul style="list-style-type: none"> • the top-down approach to solution development • refinement of a proposed solution • modification of an existing solution <p>Data types</p> <ul style="list-style-type: none"> • data types used in solutions, including: <ul style="list-style-type: none"> – integer – string – floating point – boolean – date and currency format • data structures, including: <ul style="list-style-type: none"> – 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 <p>Structured algorithms</p> <ul style="list-style-type: none"> • methods for representing algorithms: <ul style="list-style-type: none"> – pseudocode – flowcharts • control structures: <ul style="list-style-type: none"> – sequence – selection (binary, multiway) – iteration (pre-test, post-test), including: for ... next loops • software structure <ul style="list-style-type: none"> – subroutines – modularity • use of standard algorithms, including: <ul style="list-style-type: none"> – load and print an array – process records from a sequential file • checking the algorithm for errors • historical events that led to the development of a structured approach to algorithm design 	<ul style="list-style-type: none"> • develop a systematic approach to the development of a solution • select the most appropriate data type for the solution to a particular problem and discuss the merit of the chosen type • interpret and create algorithms represented in both pseudocode and flowcharts • identify control structures in an algorithm • detect logic errors in an algorithm by performing a desk check • 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:
<p>Coding in an approved programming language</p> <ul style="list-style-type: none"> • meta-languages, including: <ul style="list-style-type: none"> – BNF – EBNF – Railroad diagrams • reading and writing statements in meta-languages • the syntax of the statements used to represent the control structures, including: <ul style="list-style-type: none"> – sequence – selection (binary, multiway) – iteration (pre-test, post-test) – combinations of these • the syntax of the statements used to define and use a range of data types, including: <ul style="list-style-type: none"> – integer – string – floating point – one-dimensional array – record – sequential files <p>Error correction techniques</p> <ul style="list-style-type: none"> • types of coding errors, including: <ul style="list-style-type: none"> – syntax errors – runtime errors – logic errors • stubs <ul style="list-style-type: none"> – used to check the connection between modules of code • flags <ul style="list-style-type: none"> – 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 • debugging output statements: <ul style="list-style-type: none"> – 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 	<ul style="list-style-type: none"> • use meta-language statements from manuals and help files to develop syntactically correct code • verify the syntax of a command using meta-language statements • generate appropriate source code by: <ul style="list-style-type: none"> – using a programming environment to generate and execute code – coding an algorithm into the chosen programming language – using different data types in solutions • trace the output of a given code fragment and modify it appropriately • run, correct and extend existing code • 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 • use automated debugging features in programming environments

Students learn about:	Students learn to:
<p>Libraries of code</p> <ul style="list-style-type: none"> • reusable code <ul style="list-style-type: none"> – standard routines, such as data validation, date conversion and words to numbers • combining code and modules from different sources <ul style="list-style-type: none"> – copying and pasting into code – ways of calling modules of code – sharing/passing variables between modules <p>User interface development</p> <ul style="list-style-type: none"> • consult with users • the different perspectives a user and a developer have to a program • effective user interfaces <ul style="list-style-type: none"> – 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 <p>Documentation</p> <ul style="list-style-type: none"> • types of documentation <ul style="list-style-type: none"> – documentation for developers – documentation for users • internal documentation <ul style="list-style-type: none"> – meaningful variable names (intrinsic) – readability of code (comments, white space and indentation) • online help 	<ul style="list-style-type: none"> • 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:
<p>Test data</p> <ul style="list-style-type: none"> • selecting data for which the expected output is known • the need for thorough test data • the selection of appropriate test data, including: <ul style="list-style-type: none"> – 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: <ul style="list-style-type: none"> – desk checking an algorithm – stepping through a coded solution line by line 	<ul style="list-style-type: none"> • 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:
<p>Evaluation of design</p> <ul style="list-style-type: none">• comparing different solutions to the same problem<ul style="list-style-type: none">– different interpretations of the design specifications– the advantages and disadvantages of different approaches to reaching the solution• peer checking• structured walk through• desk checking <p>Evaluation of implemented solution</p> <ul style="list-style-type: none">• checking the solution to see if it meets the original design specifications• user feedback• social and ethical perspective	<ul style="list-style-type: none">• 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

Students learn about:	Students learn to:
<ul style="list-style-type: none">• 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 <p>Interpretation</p> <ul style="list-style-type: none">• reading original documentation in order to understand the code<ul style="list-style-type: none">– documents for the user (including user manuals)– documents for software developers• reading original algorithms to identify:<ul style="list-style-type: none">– inputs to the algorithm– the types of variables used– processes used– outputs• creating algorithms for source code when they are not available <p>Documentation</p> <ul style="list-style-type: none">• using supplied documentation to:<ul style="list-style-type: none">– identify the control structures that have been used– explain how variables have been used	<ul style="list-style-type: none">• 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

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:
<p>Implementing projects the steps in implementing project(s) include:</p> <ul style="list-style-type: none"> • defining the problem <ul style="list-style-type: none"> – understanding the problem – identification of inputs, processes and outputs to be applied to the problem • planning <ul style="list-style-type: none"> – 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:	Students learn to:
<ul style="list-style-type: none"> • building <ul style="list-style-type: none"> – 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 <ul style="list-style-type: none"> – testing of the solution using test data – evaluation of the completed solutions • modifying <ul style="list-style-type: none"> – changing the solution to meet the specifications <p>Project management techniques</p> <ul style="list-style-type: none"> • identification of tasks • identification of techniques to assist project management, including: <ul style="list-style-type: none"> – 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 <p>Project documentation</p> <ul style="list-style-type: none"> • relevant documentation may include the use of: <ul style="list-style-type: none"> – algorithms – Gantt charts – manuals – systems documentation – data dictionaries – diaries – CASE-tools <p>Social and ethical issues related to project work</p> <ul style="list-style-type: none"> • relevant issues may include: <ul style="list-style-type: none"> – ease of use – gender bias – accessibility of technical language – copyright – ergonomics 	<ul style="list-style-type: none"> • 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:
<p>Rights and responsibilities of software developers</p> <ul style="list-style-type: none"> • authorship • reliability • quality • response to problems • code of conduct • viruses <p>Software piracy and copyright</p> <ul style="list-style-type: none"> • concepts associated with piracy and copyright, including: <ul style="list-style-type: none"> – intellectual property – plagiarism – shareware – public domain – ownership versus licensing – copyright laws – reverse/backwards engineering – decompilation – licence conditions – network use • various national perspectives to software piracy and copyright laws • the relationship between copyright laws and software license agreements <p>The software market</p> <ul style="list-style-type: none"> • maintaining market position • the effect on the marketplace <p>Significant social and ethical issues</p> <ul style="list-style-type: none"> • national and international legal action resulting from software development • public issues, including: <ul style="list-style-type: none"> – the year 2000 problem – computer viruses – reliance on software 	<ul style="list-style-type: none"> • identify the impact on consumers of inappropriately developed software • interpret copyright agreements and develop personal practices that reflect current laws • acknowledge all sources in recognition of the intellectual contribution of authors • 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:
<p>Software development approaches</p> <ul style="list-style-type: none"> • approaches used in commercial systems, including: <ul style="list-style-type: none"> – the structured approach – prototyping – rapid applications development – end user development – combinations of any of the above • methods of implementation <ul style="list-style-type: none"> – direct cut over – parallel – phased – pilot • current trends in software development, for example: <ul style="list-style-type: none"> – 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 <ul style="list-style-type: none"> – software versions – data dictionary – test data – production of documentation 	<ul style="list-style-type: none"> • 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: <ul style="list-style-type: none"> – 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:
<p>Defining the problem</p> <ul style="list-style-type: none"> • identifying the problem <ul style="list-style-type: none"> – needs – objectives – boundaries • determining the feasibility of the solution <ul style="list-style-type: none"> – is it worth solving? – constraints – budgetary – operational – technical – scheduling – possible alternatives – social and ethical considerations <p>Design specifications</p> <ul style="list-style-type: none"> • the developer's perspective in consideration of: <ul style="list-style-type: none"> – data types – algorithms – variables • the user's perspective <p>Modelling</p> <ul style="list-style-type: none"> • representing a system using diagrams, including: <ul style="list-style-type: none"> – Input Process Output (IPO) diagrams – story boards – data flow diagrams – systems flowcharts – screen designs – consideration of use of a limited prototype <p>Communication issues, including:</p> <ul style="list-style-type: none"> • the need to empower the user • the need to acknowledge the user's perspective • enabling and accepting feedback 	<ul style="list-style-type: none"> • develop and interpret design specifications from a user's perspective, considering: <ul style="list-style-type: none"> – 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 • evaluate the extent to which a proposed system will meet user needs <ul style="list-style-type: none"> • 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 <ul style="list-style-type: none"> • 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:
<p>Customisation of existing software solutions</p> <ul style="list-style-type: none">• identification of relevant products• customisation• cost effectiveness <p>Documentation of the overall software solution</p> <ul style="list-style-type: none">• tools for representing a complex software solution include:<ul style="list-style-type: none">– algorithm descriptions– system flowcharts– structure diagrams– data flow diagrams– data dictionary <p>Selection of language to be used</p> <ul style="list-style-type: none">• event-driven software<ul style="list-style-type: none">– driven by the user– program logic• sequential approach<ul style="list-style-type: none">– defined by the programmer• relevant language features• hardware ramifications• Graphical User Interface (GUI)	<ul style="list-style-type: none">• evaluate the effectiveness of using commercially developed software• 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

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:
<p>Interface design in software solutions</p> <ul style="list-style-type: none"> • the design of individual screens, including: <ul style="list-style-type: none"> – identification of data required – current popular approaches – design of help screens – audience identification – consistency in approach <p>Language syntax required for software solutions</p> <ul style="list-style-type: none"> • 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: <ul style="list-style-type: none"> – multi-dimensional arrays – arrays of records – files (sequential and relative/random) – random number generators <p>The role of the CPU in the operation of software</p> <ul style="list-style-type: none"> • machine code and CPU operation <ul style="list-style-type: none"> – 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 <p>Translation methods in software solutions</p> <ul style="list-style-type: none"> • different methods include: <ul style="list-style-type: none"> – compilation – incremental compilation – interpretation • the translation process • advantages and disadvantages of each method 	<ul style="list-style-type: none"> • 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 • utilise the correct syntax for new commands using the metalanguage specification • produce syntactically correct statements • implement a solution utilising a complex algorithm • recognise and interpret machine code instructions • 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:
<p>Program development techniques in software solutions</p> <ul style="list-style-type: none"> • structured approach to a complex solution, including: <ul style="list-style-type: none"> – 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: <ul style="list-style-type: none"> – syntax errors – logic errors – peer checking – desk checking – use of expected output – run time errors, including: <ul style="list-style-type: none"> - arithmetic overflow - division by zero - accessing inappropriate memory locations • the use of software debugging tools, including: <ul style="list-style-type: none"> – use of breakpoints – resetting variable contents – program traces – single line stepping <p>Documentation of a software solution</p> <ul style="list-style-type: none"> • forms of documentation, including: <ul style="list-style-type: none"> – 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 <ul style="list-style-type: none"> – use of CASE tools 	<ul style="list-style-type: none"> • 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 <ul style="list-style-type: none"> • produce user documentation (utilising screen dumps) that includes: <ul style="list-style-type: none"> – 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:
<p>Hardware environment to enable implementation of the software solution</p> <ul style="list-style-type: none">• hardware requirements<ul style="list-style-type: none">– minimum configuration– possible additional hardware– appropriate drivers or extensions <p>Emerging technologies</p> <ul style="list-style-type: none">• hardware• software• their effect on:<ul style="list-style-type: none">– human environment– development process	<ul style="list-style-type: none">• recognise the need for additional hardware • 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.

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:
<p>Modification of code to meet changed requirements</p> <ul style="list-style-type: none">• 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 <p>Documentation of changes</p> <ul style="list-style-type: none">• source code, macro and script documentation• modification of associated hard copy documentation and online help• use of CASE tools to monitor changes and versions	<ul style="list-style-type: none">• read and interpret others' code, macros and scripts• design, implement and test modifications• recognise the cyclical approach to maintenance • 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:
<p>Designing and developing a software solution to a complex problem</p> <p>Defining the problem and its solution, including:</p> <ul style="list-style-type: none"> • defining the problem <ul style="list-style-type: none"> – identification of the problem – idea generation – communication with others involved in the proposed system • understanding <ul style="list-style-type: none"> – 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 <ul style="list-style-type: none"> – 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 	<ul style="list-style-type: none"> • 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:
<p>Systems implementation</p> <p>Implementing the software solution by:</p> <ul style="list-style-type: none"> • implementation <ul style="list-style-type: none"> – 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 <ul style="list-style-type: none"> – modifying the project to ensure an improved solution 	<ul style="list-style-type: none"> • 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:
<p>Historical reasons for the development of the different paradigms</p> <ul style="list-style-type: none"> • 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 <p>Basic building blocks</p> <ul style="list-style-type: none"> • variables and control structures (imperative) • functions (functional) • facts and rules (logic) • objects, with data and methods or operations (object oriented) <p>Effect on programmers' productivity</p> <ul style="list-style-type: none"> • speed of code generation • approach to testing • effect on maintenance • efficiency of solution once coded • learning curve (training required) <p>Paradigm specific concepts</p> <ul style="list-style-type: none"> • logic paradigm <ul style="list-style-type: none"> – (eg Prolog, expert system shells) – heuristics – goal – inference engine – backward/forward chaining • object oriented programming <ul style="list-style-type: none"> – (eg C++, Delphi, Java) – methods – classes – inheritance – polymorphism – encapsulation – abstraction • functional (eg LISP, APL) <ul style="list-style-type: none"> – functions 	<ul style="list-style-type: none"> • 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:
<p>Representation of data within the computer</p> <ul style="list-style-type: none"> • character representation, namely: <ul style="list-style-type: none"> – ASCII – hexadecimal • integer representation, including: <ul style="list-style-type: none"> – sign and modulus – one’s complement – two’s complement • representation of fractions, namely: <ul style="list-style-type: none"> – floating point or real • binary arithmetic, including: <ul style="list-style-type: none"> – addition – subtraction using two’s complement representation – multiplication, shift and add – division, shift and subtract 	<ul style="list-style-type: none"> • 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:
<p>Electronic circuits to perform standard software operations</p> <ul style="list-style-type: none"> • logic gates, including: <ul style="list-style-type: none"> – AND, OR, NOT, NAND, NOR, XOR • truth tables • circuit design steps <ul style="list-style-type: none"> – identify inputs and outputs – identify required components – check solution with a truth table – evaluate the circuit design • specialty circuits, including: <ul style="list-style-type: none"> – half adder – full adder – flip-flops as a memory store <p>Programming of hardware devices</p> <ul style="list-style-type: none"> • the input data stream from sensor and other devices <ul style="list-style-type: none"> – header information – data characters – trailer information – control characters – hardware specifications – documentation • processing of data stream <ul style="list-style-type: none"> – the need to recognise and strip control characters – counting the data characters – extracting the data • generating output to an appropriate output device <ul style="list-style-type: none"> – required header information – required control characters – data – required trailer information • control systems <ul style="list-style-type: none"> – responding to sensor information – specifying motor operations 	<ul style="list-style-type: none"> • 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 <ul style="list-style-type: none"> • 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

- | | |
|---|--|
| <ul style="list-style-type: none">• printer operation<ul style="list-style-type: none">– control characters for features, including page throw, font change, line spacing• specialist devices with digital input and/or output | |
|---|--|

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

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

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

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

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	A process in which an application is developed by users who

development	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