HSC Software Design and Development

Major Programming Assignment

Topics:	Software Development Cycle Developing a Solution Package
Weight:	70%
Due Dates:	Staggered between December 2018 and June 2019

Contents

It is important that before you commence this project you understand how Stage 3 will be marked. You should select a project that you will find challenging, but not one that is too difficult for you to successfully complete. If you have any questions about the Stage 3 marking please speak with me.

Also, be aware that students who do not have three suitable subroutines/functions will be unable to access the top mark range in Stage 1/2 and Stage 4.

Outcomes Assessed

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

BLANK PAGE

General Points

- This is an individual project.
- All work must be your own. If any of your work is copied (either from another member of the class, or from a third party such as a book or the internet) you will receive zero. I will be actively checking sources of assignment materials.
- **Work must be submitted by 8:45am on each due date.** Work handed in late will receive zero. It will still need to be handed in to ensure that you meet the requirements of the course.
- It is your responsibility to keep backups of your work. No provisions will be made for people who lose disks, or have any other form of electronic misadventure.
- It is recommended you use Google Drive to store your working copy of your project. Apart from handling backup, it also means it is always accessible at school during project lessons.
- Approximately one lesson per cycle will be allocated to this project.

Task Specifications

Task specifications should be read in conjunction with the marking criteria.

You are required to design, develop and document a substantial piece of software. You are free to develop a piece of software that is of interest to you, **subject to teacher approval. This approval must be sought prior to Friday 9 November 2018.**

If you choose to create a game for your project then it must meet these two conditions:

- it must be based on a pre-existing game. You may not create/devise your own custom game (e.g. role-playing game).
- it must be a single game, not a suite of games or unrelated mini-games.

The minimum requirements are:

- The software should read from and write to a file.
- The software should utilise data structures (i.e. arrays or records).
- A sorting algorithm must be used. It must be a correct implementation of a syllabus sort (bubble, selection or insertion).
- At least one subroutine[1] and at least two functions[2] must be used. **These are to be developed by you**, and must accept data in the form of parameters. They must not interact with the interface. Any subs or functions in Appendix E can't be used to meet this minimum requirement.
- One of the functions must be a player name validation. It must accept a string for the player name and return a Boolean indicating whether the player name is valid. You may determine the validation rule(s).

The project must be completed in Microsoft Visual Basic 2010 Express Edition.

You must also keep a logbook, and keep it up to date. The logbook must be completed on the template provided in Google Drive.

A few words about asking me for help, particularly in the area of programming. This project should be substantially your own work. There is nothing wrong with asking for help when you are stuck. Where this becomes an issue is if the only time **any** progress gets made on your project is when I am helping you with it. If this is the case, it will adversely affect your Stage 3 mark. I will give you warning if this is the case.

---

[1] If a series of instructions are to be used many times in other sections of a program, it's possible to group these instructions together as a subroutine. Other portions of the program can execute the batch of instructions within the subroutine by merely calling that subroutine. Subroutines are identified by names and can often accept parameters to make their functions more generic. Reference: http://www.une.edu.ve/administracion/virtuabook/activex/paxxb.htm

[2] A function is a special type of subroutine that returns a single value. This is performed by setting the function name to the return value or using a RETURN statement.

Emphasis of the project

A very important concept in this project is a modular approach to programming. This means that you should break down the program into subroutines. Each subroutine should form one logical task.

When planning your project, consider this advice:

- Break down your program into small parts.
- Use a structure chart.
- Minimise the use of global variables. Wherever possible, pass your variables as parameters.
- Know what a function is and when to use one.
- You will have a number of subs and functions that will perform processing of data. Examples may be a CheckWinner subroutine, validation routine for a username, or a sorting algorithm for high scores. These should not interact with the user interface (i.e. they should contain no references to forms or controls). Apart from being good practice, you won't be able to properly test your subs and functions if they interact with the UI.
- It is imperative that you read and understand the task description, submission instructions, and marking criteria for each stage as these will be used to assess your work.

Students who ignore this advice will lose substantial marks at every stage of the project.

Examples of projects

Examples of appropriate assignments include, but are not limited to:

- Stock market simulator
- Calculator
- Appointment system
- Hotel bookings
- Subject choice/report system
- Payroll system
- Cloze test
- Stock control
- Batch banking/ audit system
- Text encryption/ analysis
- Olympic ticketing
- Library system
- Connect four
- Mastermind
- Snakes and Ladders
- Hangman
- Clock Patience
- Memory
- Battleship
- Yahtzee
- Draughts
- Search a Word
- Nine letter word puzzle
- Monopoly
- Bingo
- Ludo
- Guess Who
- Cluedo
- Sorry
- Trouble
- Crossword generator
- Quizzes
- Drill and practice games
- Database simulators
- Assembler
- Sample word processor
- Athletics carnival
- Resource allocation
- Cash register
- Tenpin bowling scoring
- Calculation of sport averages
- Wedding planner

Ownership of projects

You retain intellectual property rights over your projects.

CGS staff will:

- Keep copies of your project. These copies may be electronic, paper, or both.
- Use your work samples for the purposes of Registration.

CGS staff may:

- Show samples of your work to future cohorts.

Tasks and Due Dates

The task details should be read in conjunction with the marking criteria.

SDC Stage	Task	Submission Instructions	Due Date *

Stage 1 and 2: Understanding the problem/Planning and designing software solutions
 	
Outcomes: H3.1, H3.2, H4.2, H4.3, H5.2, H5.3, H6.4
 	
Weight: 20% of HSC Assessment
	• Demonstrate your understanding of the problem through a report. The report should be in three sections, as outlined below.
	• The structure of your report should be:
	  • **Design Specifications**
	  • A list of design specifications. This should be a list of measurable specifications you hope your program will achieve. Design specifications can be from both the user's and developer's perspective. See Appendix B for sample Design Specifications.
	  • **User section**
	  • Overview of the program
	  • Screen designs (Draw.IO is recommended. Use the *mockups* group of tools)
	  • Description of the screen designs from the user's perspective.
	  • **Developer section**
	  • Level 1 Data flow diagram. (Draw.IO or EDraw Max are two possible programs to use)
	  • Data structures that will be used. See Appendix B for more detail on this.
	  • A structure chart (i.e. structure diagram). This will be accompanied by a description of the purpose of each module, which includes reference to the *control structures* used in the module. The structure chart is complete and shows all necessary sub-tasks and with an appropriate degree of decomposition and modularity.
	  • For two functions and one subroutine (one of which is your sorting algorithm), provide:
	  • IPO diagram
	  • Data dictionary. See Appendix B for the required format.
	  • Pseudocode for the two functions and one subroutine. Visual Basic code is not acceptable.
	  Note that the two functions and one subroutine must be independent of the interface.
		• Typed in single Word document. Contains an automatically generated table of contents, automatic page numbering, page breaks where appropriate, and uses heading styles. All parts of the document must be entirely computer generated. This also means that scans of handwritten work is **not** permitted.
		• Hard copy submitted. No plastic sleeves, staple in the top left corner only.
		• Electronic docx copy submitted via Google Classroom
		File name convention is:
		`Last First Stage 1 and 2`
		For example:
		`Smith John Stage 1 and 2.docx`
			8:45am Friday 7 December 2018
			
			Term 4 Week 8

	• Logbook. See Appendix A. All correspondence with the teacher about your project should be summarised in your logbook.	• In the template provided in Google Drive. Do not modify it between the submission deadline and when permission is given for changes to be made.	

SDC Stage	Task	Submission Instructions	Due Date *
Stage 3: Implementing software solutions Outcomes: H4.2, H4.3, H5.1, H5.2, H5.3, H6.2, H6.3 Weight: 40% of HSC Assessment	• Code the solution.	• In a ZIP file uploaded to Google Classroom. The ZIP file must be called: `Last First Stage 3` e.g. `Smith John Stage 3` • Must include all source code files. • Compiled exe or install files should also be provided. • **Regardless of the format in which you present Stage 3 you must provide a copy of all files in your ZIP file.**	**8:45am Friday 24 May 2019** Term 2 Week 4
	• Document this process in a logbook particularly noting how the code was developed and how errors were identified and corrected (an example of a suggested format/style for a logbook is contained in Appendix A). All correspondence with the teacher about your project should be summarised in your logbook.	• In the template provided in Google Drive. The logbook must not be modified between the submission deadline and when permission is given for changes to be made.	
	• Create technical and user documentation for the solution. These should be presented in an appropriate format. A complete structure chart of the solution. Note: Examples of a user guide and a tutorial can be found in the project work folder in Google Drive . These can be used as examples of how to structure and present the user documentation. If you wish to have your user manual printed and bound in A5 booklet format, email it to me in A4 pdf format by 10:00pm on Wednesday 22 May 2019. The number of pages should be a multiple of 4 and it is recommended that the font size be 16 point.	• Electronic resources (e.g. tutorial, help files) must be provided in your Stage 3 ZIP file. • Electronic copies of printed resources must also be placed in your Stage 3 ZIP file. • For the structure chart it should be provided in an electronic format (pdf recommended) in your `Last First Stage 3` ZIP file.	

Stage 3 is challenging. Ensure you make a good start on it during the April holidays.

If you find yourself in a panic about the project, particularly as the deadline approaches, make sure you speak with your teacher about your concerns. We will do our best to support you, and may be able to provide advice about what you can do to maximise your marks.

SDC Stage	Task	Submission Instructions	Due Date *

Stage 4: Testing and evaluation of software solution
 	
Outcomes: H4.2
 	
Weight: 10% of HSC Assessment
	• Test the solution and document the testing procedure. This will include testing of the three nominated subroutines. See Appendix D for how this should be set out.
	
	As part of testing, you should describe any crashes that occur in your program, including steps that lead to the crash, the cause of the error, and suggestions on how it could be fixed.
	• Evaluate the finished product using:
	  • The list of original design specifications written in stage 1 (this should not be limited to original screen designs).
	  • User evaluations (surveys, interview if desired). Surveys must be online using a Google Form. Surveys should focus on ease of use, the user interface, help file, tutorial etc. There should be a combination of closed and open questions.
	
	An Excel template to assist you with the presentation of graphs will be provided.
		• Typed in single Word document. Contains an automatically generated table of contents, automatic page numbering, page breaks where appropriate, and uses heading styles. All parts of the document must be entirely computer generated. This also means that scans of handwritten work is **not** permitted.
		• Hard copy submitted. No plastic sleeves, corner staple only.
		• Electronic docx copy submitted via Google Drive.
		
		File name convention is:
		`Last First Stage 4`
		For example:
		`Smith John Stage 4.docx`
			8:45am Friday 28 June 2019
			
			Term 2 Week 9

	• Document the testing and evaluation process in the logbook. The logbook for Stage 4 will typically be more brief than previous stages. It should contain a log of the work completed, including survey development, the survey process, testing, and any issues that have arisen in this process.	• In the template provided in Google Drive. The logbook must not be modified after the submission deadline.	

** These dates have been set before the 2019 calendar has been published. It may be necessary to change one of these dates to avoid a clash. e.g a pupil free day or another assessment task due on the same day. If this is the case, the due date will be brought forward, and you will be given at least two weeks notice, in writing or via email, of the change of date.*

Marking Criteria

Stage 1 and 2: Understanding the problem, Planning and designing software solutions

Criteria	Mark

• Report is divided into three main sections.
• The Design Specifications should be a list, each item should be measurable, and the Specifications contain items from both the user's and developer's perspective.
• User's section should contain preliminary screen designs to demonstrate the entire user interface. The diagrams should be accompanied by a written explanation of the intended design.
• The developer's section should consist of:
  • a data flow diagram of the intended system. This should be accompanied by a description of the data flow diagram.
  • a summary of the data structures (files, arrays, records etc) that will be used. This is in accordance with Appendix B.
  • a complete structure chart which demonstrates an appropriate level of decomposition and modularity (i.e. contains one logical task per subroutine). This should be accompanied by a description of each module in the structure chart. The description of each module should state its purpose and make reference to the *control structures* used in the module.
  • For your chosen subroutine and two functions:
  • a complete IPO diagram which provides detail of all processes performed by the module.
  • a correctly formatted data dictionary describing all of the variables to be used. The scope of all variables is indicated. The data dictionary is in the format prescribed in Appendix B.
  • a detailed set of efficient algorithms describing each module. This is provided in pseudocode form that conforms to the NESA *Software and Course Specifications* document. Each module has been designed with an appropriate level of decomposition and modularity.
One of the subroutines or the function is the sorting algorithm. It must be a correct implementation of a syllabus sorting algorithm.
• The report is free from spelling and grammatical errors.
• The report is professionally presented, using automatic table of contents, automatic page numbering, page breaks where appropriate, heading styles, and is submitted in both hard copy and in docx format. The report is contained in a single Word document.
 	
To achieve a mark in this range, students must also:
• present screen designs created using appropriate computer drawing tools such as Draw.IO. (you must not use screenshots from Visual Basic).
• address, in detail, each of the minimum project requirements as outlined on page 4. This includes detail about each of the subs and functions that will be used.
• present a logbook using the Google Drive template which is of a similar style to Appendix A. All logbook entries are made in a timely manner. This means that the dates next to each entry reflect the date that the entry was made, and entries were not predominantly made close to the deadline. The first entry addresses backup strategies. All teacher correspondence is included.
• use correct judgement in dividing material between the user and developer sections.
• Provide consistency between the different components of their submission. For example, if a player name field appears in the screen designs, it must also be in the structure chart.
	13 – 15

• A report, divided into three sections, defining the design specifications, accompanied by preliminary screen designs, data flow diagram, structure chart and summary of variables and data structures. • Correct pseudocode and IPO diagram for at least one subroutine. • A logbook in Google Drive which follows the style of Appendix A.	8 – 12
• A report defining the problem accompanied by a subset of the requirements listed above.	3 – 7
• A brief description of the problem without any of the required charts and diagrams.	1 – 2

Students who do submit all the required components of Stage 1 and 2 but do not completely meet the submission instructions will have one mark deducted from their Stage 1 and 2 mark. For example, if your mark against the criteria above is 10/15, one mark is deducted and you will receive 9/15.

Stage 3: Implementing software solutions

Criteria	Mark

A fully working solution that includes all of:
• (A) a fully working set of source code files
• (B) thorough internal documentation. This includes the use of one logical task per subroutine, explanatory comments provided in all subroutines and functions, appropriate use of spacing in the code. XML comments on each subroutine/function are provided in accordance with Appendix C.
• (C) thorough intrinsic documentation. This includes all variable names, names of control structures, all controls such as labels and buttons, and forms. Adherence to naming conventions such as frm, lbl etc is also examined here.
• (D) a structure chart. It should be complete, and consistent with the code. Symbols and structure chart elements are used correctly.
• (E) a set of appropriately and professionally presented user documentation including
  • user guide[5] (incorporating hardware specifications, installation guide, user manual, troubleshooting guide)
  • tutorial to introduce new users to the software. Note that the tutorial should be interactive.
• (F) a logbook detailing the development of the project. The logbook addresses:
  • new things learnt each day/week
  • any difficulties encountered and how they were overcome
  • how errors were identified and corrected
  • any changes made to the original design and why they were made
  • all correspondence with your teacher about your project.
All logbook entries are made in a timely manner. This means that the dates next to each entry reflect the date that the entry was made, and entries were not predominantly made close to the deadline.
 	
In addition, projects in this mark range will:
• Read from and write to a file
• Use a working sorting algorithm. It must be a correct implementation of a syllabus sorting algorithm (bubble, selection or insertion)
• Utilise data structures (arrays or records)
• Use at least one subroutine. The sub will be user-developed and must accept data in the form of parameters. It must not interact with the interface.
• Use at least two functions. The function will be user-developed and must accept data in the form of parameters. A single value is returned by setting the name of the function to the return value. The functions do not interact with the interface.
• One of the functions is a player name validation function. It accepts a player name as a string and returns a Boolean. It does not interact with the interface.
• Be designed and coded with modules which demonstrate an appropriate level of decomposition and modularity.
• Have three easily identifiable subroutines/functions for testing in Stage 4. They will perform a single task, receive data as parameters and be independent of the interface.
• Use functions appropriately to minimise the repetition of code.
• Use variable scope appropriately, i.e. variables should be local whenever possible.
• Use control structures appropriately, for example using multiway selection in preference to long If-Elseif-Else-EndIf statements.
• Declare constants using the const command where necessary.
• Avoid repetitive blocks of code wherever possible. This can usually be achieved through the use of data structures such as arrays, as well as control arrays on forms.
• Use data structures appropriately to store data about a program. For example, in a board game, information about objects on a board should be primarily held in an array, not in the text properties of the form elements.
• Use code which is faithful to the course requirements. Use of exit or goto statements will not meet this criteria.
• Be professionally presented, to the same standard as a piece of commercial software. This includes providing work free from spelling and grammatical errors.
• Not crash, provide unusual error messages, or behave in an unexpected manner which adversely affects the workflow (or gameplay, in the case of a game) of the program.
	13 – 15

[5] The user guide should be a printed document and contain screenshots. A doc, docx or pdf copy should be provided as well as the printout.

• A solution that runs that includes most of the elements (A), (B), (C), (D), and (E) above.
 	
In addition, projects in this mark range will:
• Read from and write to a file
• Use a working sorting algorithm. It must be a correct implementation of a syllabus sorting algorithm (bubble, selection or insertion)
• Utilise data structures (arrays or records)
• Use at least one subroutine. The sub will be user-developed and must accept data in the form of parameters. It must not interact with the interface.
• Use at least one function. The function will be user-developed and must accept data in the form of parameters. A single value is returned by setting the name of the function to the return value. The function does not interact with the interface.
• Includes a player name validation function. It accepts a player name as a string and returns a Boolean. It does not interact with the interface.
• Have a logbook detailing the development of the project. The logbook addresses most of the following points:
  • a summary of your progress
  • new things learnt each day/week
  • any difficulties encountered and how they were overcome
  • how errors were identified and corrected
  • any changes made to the original design and why they were made
  • all correspondence with your teacher about your project.
	9 – 12

• A non-working solution that contains all of the documentation above OR • A working solution that contains a subset of the documents above.	5 – 8
• A non-working solution that is accompanied by a subset of the documents above.	1 – 4
• A submission that does not provide source code, regardless of the quality of the project or the accompanying documentation OR • A submission that is plagiarised.	0

Important information for Stage 3:

- Projects must be completed in Visual Basic 2010 Express Edition.
- Any code or part of the program developed outside Visual Basic 2010 Express Edition will not be marked. It will not be considered as part of discretionary mark allocation. The treatment of non-Visual Basic code will be the same as if you are calling an external library.
- The projects will be assessed on a computer running the latest version of Windows 10 as at the submission date.

Students who do submit all the required components of Stage 3 but do not completely meet the submission instructions will have one mark deducted from their Stage 3 mark. For example, if your mark against the criteria above is 7/15, one mark is deducted and you will receive 6/15.

Note: There are an additional 5 marks in this part of the assignment. (i.e. each student will receive a mark out of 20 for this stage). They are allocated according to teacher discretion and reflect the complexity of the project and the professionalism of the overall presentation and product. **These marks are primarily used to differentiate between projects of different complexity**, but also to reward students who produce outstanding work. The marks are used to rank students appropriately. If you require constant help to write your program, this will adversely affect your mark in this section.

The primary consideration for the allocation of these marks is evidence in the logbook and the source code of complex and innovative coding. Additionally, if you implement complex algorithms (e.g. an AI-based computer player) it is expected that you provide sufficient documented explanation of how it works (both conceptually and technically) in order for me to understand it and correctly assess the work you have completed. The most appropriate location for such explanation is likely to be in your journal, and we encourage the use of diagrams (and similar supporting material) if needed to assist with any explanations. Remember, the clearer that your explanation of complex algorithm design is means that I can better understand what you have done and award marks for your work.

In past years, students with poor logbooks, no internal documentation, not nearly enough intrinsic documentation and very poor setting out of their code have felt that they were unfairly assessed for the additional ten marks. You bear the responsibility here to make sure I can assess the complexity of your program properly.

Examples of projects from past years and the additional marks that they have scored. Note that in previous years discretionary marks have been out of 10. In 2019 they will be out of 5.

2010 and earlier complexity marks Scrabble: 9 or 10 Monopoly: 7 Roulette: 7 Mastermind: 7 Connect Four: 5 Snake: 5 Guess Who: 2 or 3 Snakes and Ladders: 2	**Class of 2011 complexity marks** Universal Translator: 10 Battleship: 7 – 10 Tetris: 8 Snake: 7 Connect Four: 5 – 7
Class of 2012 complexity marks Minesweeper: 9 Risk: 7 Connect Four: 5 Guess Who: 3 Bingo: 2	**Class of 2013 complexity marks** Uno: 10 Pong: 9 Hold 'em Poker: 9 Connect Four: 5 – 7
Class of 2014 complexity marks Ghosts: 10 Air Traffic Control Simulation: 9 Chess: 9 Reversi: 8 Ludo: 8 Checkers: 5 Nine letter word game: 5 Dots and Boxes: 5 Connect Four: 5	**Class of 2015 complexity marks** Tron: 10 Bomberman: 9 Pacman: 8 Phalanx: 8 Five Hundred: 7 Tank Wars: 6 Connect Four: 5 – 7
Class of 2016 complexity marks Tetris: 10 Minesweeper: 9 Scrabble: 9 Backgammon: 8 Battleship: 5 Sudoku: 4 Nine letter word game: 2	**Class of 2017 complexity marks** Maze puzzle: 10 Tetris: 9, 7 Boggle: 9 Scrabble: 7 Minesweeper: 7 Reversi: 5 Battleship: 4

This is a guide only. It doesn't take into consideration the complexity of the overall product, nor professionalism. For example, a game such as Connect 4 can be done in a complex manner or a very simple manner. Two-player Connect 4 is easier to create than one-player, and there is potential for complex work when designing the computer's game algorithm.

The additional marks are used to differentiate projects within a cohort (e.g. class of 2017), not between students over different years. For example, in 2007 a student who did Roulette got 7, but in 2010 a student who did Roulette got 4.

Stage 4: Testing and evaluation of software solution

Criteria	Mark
• A report that describes the test plan, reports on the conduct of the tests and describes the test results including changes required to the program as a result of the testing procedure. Thorough testing is carried out on the three nominated subroutines. The test results are presented in a manner conforming with Appendix D. Submissions in this mark range should test the program • on computers with different specifications (must be physical hardware, not virtualised hardware) AND • using different install locations See Appendix D for how results should be presented for testing on different computers. If the program crashed then a thorough description is provided of the crash, including steps that lead to the crash, the cause of the error, and suggestions on how it could be fixed. • The report evaluates the finished product against the list of original design specifications from Stage 1. This will include a comparison of the original screen designs against the final screens. • User feedback is sought, summarised and reported. Submissions in this mark range will have well-designed surveys, will seek wide input (at least 10 respondents), and provide thorough analysis of the survey responses. Tools such as Excel will be used to summarise the survey results. Graphs will be presented in a professional manner[6]. There is evidence of reflection on the user input. A screenshot of the completed survey results should be provided as evidence of the minimum 10 respondents. A screenshot of the results google sheets spreadsheet should be provided. It must be legible. • The report is professionally presented, using automatic table of contents, automatic page numbering, page breaks where appropriate, heading styles, and is submitted in both hard copy and in docx format. The report is contained in a single Word document. To achieve a mark in this range, students must also: • present their work in a professional manner, free from spelling and grammatical errors. • present their log book in a manner conforming with Appendix A. All logbook entries are made in a timely manner. This means that the dates next to each entry reflect the date that the entry was made, and entries were not predominantly made close to the deadline.	13 – 15
• A report with a description of the test results including changes required to the program as a result of the testing procedure. Testing of at least one nominated subroutine is carried out and presented in a manner conforming with Appendix D. • Report covers evaluation of program but may not seek wide input. • Submissions which do an outstanding job on either testing or evaluation, but a poor job on the other, also fall into this mark range.	8 – 12
• A report on the conduct of the tests and a description of the test results.	3 – 7
• A brief description of the results of the testing stage.	1 – 2

Students who do submit all the required components of Stage 4 but do not completely meet the submission instructions will have one mark deducted from their Stage 4 mark. For example, if your mark against the criteria above is 10/15, one mark is deducted and you will receive 9/15.

[6] Show the survey results clearly, efficiently and meaningfully. Graphs and axes are labelled. Scales are appropriate. Wise use of colour. If graphs are printed in black and white, appropriate shading should be used so that sections of the graph are clearly distinguishable. Note that the graphs built in to the analysis tools in Google forms will not meet this criteria.

Appendix A: Sample Student Logbook

Note: Logbooks must be completed using the template in Google Drive. This appendix gives an indication of the type of content you could put in your logbook. Code fragments can be copied and pasted into the logbook.

All correspondence with your teacher about your project (email/conversations in class etc) should be summarised in your logbook. Emails should be copied and pasted into your logbook. Emails should be in a single email per project stage. A suggested email subject is "Stage 1/2 questions". The email goes back and forth between you and your teacher with all questions and answers in the one email.

Sample logbook (with thanks to Alessandro Barilaro, Class of 2016, for his permission to reproduce his work)

23/11/2015	**Accomplished** • Created preliminary structure chart • Created preliminary design document • Created preliminary screen designs • Specified subroutines and functions to elaborate on for developer section **Difficulties** • Determining suitable functions • Separating program into logical subroutines for structure chart **Learned** • How to define functions versus subroutines Began to put together final Stage 1/2 document, writing design specifications, and program overview. Also drafted screen designs digitally, and drafted structure chart on paper, scanned below in PNG format. Drafted some elements of the final Stage 1/2 document, including a DFD and design specifications. Mr Gray assisted with the structure chart, mainly on the levelling of subroutines, and what subroutines/functions should be placed where, especially within the main game loop. I had difficulty creating a structure chart, as the project requirements specify that functions must be present in the program. To remedy this, I researched how subroutines and functions differ, and came to the conclusion that several of my subroutines were closer in purpose and form to functions. All that was required was the addition of a value to be returned by each function, which was simple to do. This also made the task of selecting two functions and a subroutine to elaborate on in the design document much easier, as I now actually had functions to work with, rather than a slew of subroutines. Link to handwritten work (structure chart)

Hyperlink to relevant information (e.g. pdf of email correspondence, image of scan etc)

Logbooks should also have information about the strategies you used to locate and correct errors. Changes to the original design specifications should also be documented.

See the marking criteria for Stage 3 for further information on how your logbook will be assessed.

Appendix B: Sample Design Specifications, data dictionary, data structures, test data, Arguments to be passed, code for click events.

Design Specifications

These are examples only. Your design specifications should consider both the user's and developer's perspective.

1. The noughts-and-crosses game will operate in two modes: 2 human players, or 1 human player vs the computer.
2. Each game will be scored based on the time taken for each user to make their moves.
3. A table of 5 high scores will be available to view.
4. A timer will be used on a 500ms interval to control piece movement.

Data dictionary

Used for simple variables

Variable Name	Number of characters	Data type	Number of decimal places.	Description	Scope
PlayerName	10	String	N/A	Stores the player's name	Global
Count	N/A	Integer	N/A	Loop counter	Local to DisplayScores
WidgetBuyPrice	N/A	Currency	2	The buy price for Widgets	Local to InitialiseGame

Data Structures

Arrays

Array Name	Dimension	Index	Data type	Description	Scope
arrBoard	2	0 to 2, 0 to 2	Integer	Stores the piece type for each cell in the 3 x 3 board, where 0 = empty, 1 = blue, 2 = red.	Global
arrWeekdays	1	1 to 7	string	Stores weekdays e.g. arrWeekdays(1) = "Sunday", arrWeekdays(2) = "Monday" etc	Local to PrintPriceReport

Records

Record name: recSettings
Scope: Global

FieldName	Data Type	Description
numPlayers	Integer	Number of players in the game.
ColourScheme	Integer	Chosen Colour Scheme for Game. 1 = Blue, 2 = Pink.

Arrays of Records
Record name: recHighScores

FieldName	Data Type	Description
PName	String	Player Name
PScore	Integer	Player's score

Array of records name: arrrecHighScores
Scope: Global
Dimension: 1
Index: 0 to 5
Description Holds names and scores for top 5 players.
Note: A diagram of arrays of records is optional, but will enable you and the marker to see the structure at a glance.

Files

Filename: HighScores.txt
File location: application folder\Data\
Type of file: sequential
Structure of file: P1name
P1score
P2name
P2score
...
P5name
P5score

No sentinel value. One entry per line

Sample:

```
Adam
10
Bruce
8
Charles
7
Dave
5
Eric
1
```

Code for button click events

Best practice for thinking about button click events and your structure chart is shown by the example below.

In a game of Scissors Paper Rock, the user makes their selection of scissors, paper or rock on a form, then presses a command button titled “play”.

You do not need to put the `btnPlay` click event on your structure chart.

Rather, you should consider an algorithm such as the incomplete one below:

```
BEGIN Play
      Get UserChoice
      ComputerChoice = RandBetween(1,3)
      IF DetermineWinner(UserChoice, ComputerChoice) = 0 THEN
            PRINT “Draw”
      ENDIF
END Play
```

When this is coded in stage 3, the `btnPlay` click event code could contain one line of code, a call to the subroutine `Play`.

Appendix C: Internal Documentation Requirements

Each subroutine or function should be documented using XML. An example appears below:

```
''' <summary>
''' Function converts a string input into an integer in a specified range.
''' </summary>
''' <param name="InputString">The input string, e.g. "Hello15 World"</param>
''' <param name="Min">The minimum integer that should be returned.</param>
''' <param name="Max">The maximum integer that should be returned.</param>
''' <param name="DefaultVal">This is the integer that will be returned if the input is not valid or outside of the allowed range.</param>
''' <returns>An integer in the specified range.</returns>
''' <remarks>Written by AMG, 4 October 2014. Example of use: ValidateInteger("Hello15 World",0,20,-1) = 15.</remarks>
Public Function ValidateInteger(ByVal InputString As String, ByVal Min As Integer, ByVal Max As Integer, ByVal DefaultVal As Integer) As Integer
    Dim StringInteger As String, TempChar As String
    Dim Counter As Integer, IsADigit As Boolean

    StringInteger = ""

    'Consider each character in turn
    'If it is a digit then add it to StringInteger
    For Counter = 1 To Len(InputString)
        TempChar = Mid$(InputString, Counter, 1)
        'The ASCII code for the 10 digits range from 48 to 57
        IsADigit = Asc(TempChar) > 47 And Asc(TempChar) < 58
        If IsADigit Then
            StringInteger = StringInteger & TempChar
        End If
    Next Counter
    ValidateInteger = Val(StringInteger)

    'Check for at least some digits and that number is within the required range
    If StringInteger = "" Or ValidateInteger < Min Or ValidateInteger > Max Then
        ValidateInteger = DefaultVal
        MsgBox("A number from " & Min & " to " & Max & " is required.")
    End If

End Function
```

To create the xml skeleton for comments, click on the line *above* the function or subroutine, and type:
''' (i.e. three single quotes in a row)

Button click events can use common text for the parameters *sender* and *e*. Sample text you can use is below.

```
''' <summary>
    '''
    ''' </summary>
    ''' <param name="sender">Reference to the control which called the subroutine</param>
    ''' <param name="e">Provides more information about the event that caused this
subroutine to be called</param>
    ''' <remarks></remarks>
```

Appendix D: Testing

This page shows how you should present the testing for the nominated subroutines.

Subroutine 1: DeterminePayout

Code

Format: screenshot or copy and paste is acceptable

```
Private Function DeterminePayout(ByVal RollVal As Integer) As Integer
    'the function accepts a Roll Value (between 1 and 36). If the number is even
    'it returns a payout of $10, otherwise a payout of $3.
    If RollVal Mod 2 = 0 Then
        DeterminePayout = 10
    Else
        DeterminePayout = 3
    End If
End Function
```

Test data

Format: must be in a table with the following columns:

variable name	test data	expected output	reason for inclusion
RollVal	20	10	Testing first pathway (condition true)
RollVal	19	3	Testing second pathway (condition false)

Each row of the table represents *one set of test data*. If a set of test data includes more than one variable, it should be included in the row.

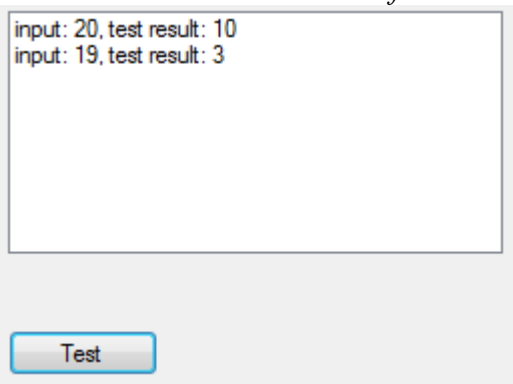
Driver code

(screenshot or copy and paste is acceptable)

```
Private Sub btnTest_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles btnTest.Click
    lstTest.Items.Add("input: 20, test result: " & DeterminePayout(20))
    lstTest.Items.Add("input: 19, test result: " & DeterminePayout(19))
End Sub
```

Test Result

Format: must be a screenshot of a listbox (not a message box, label etc)



Inputs should be shown with the outputs in the list box, like the example shown.

If the actual output differs from the expected output, then some explanation of the discrepancy should be provided.

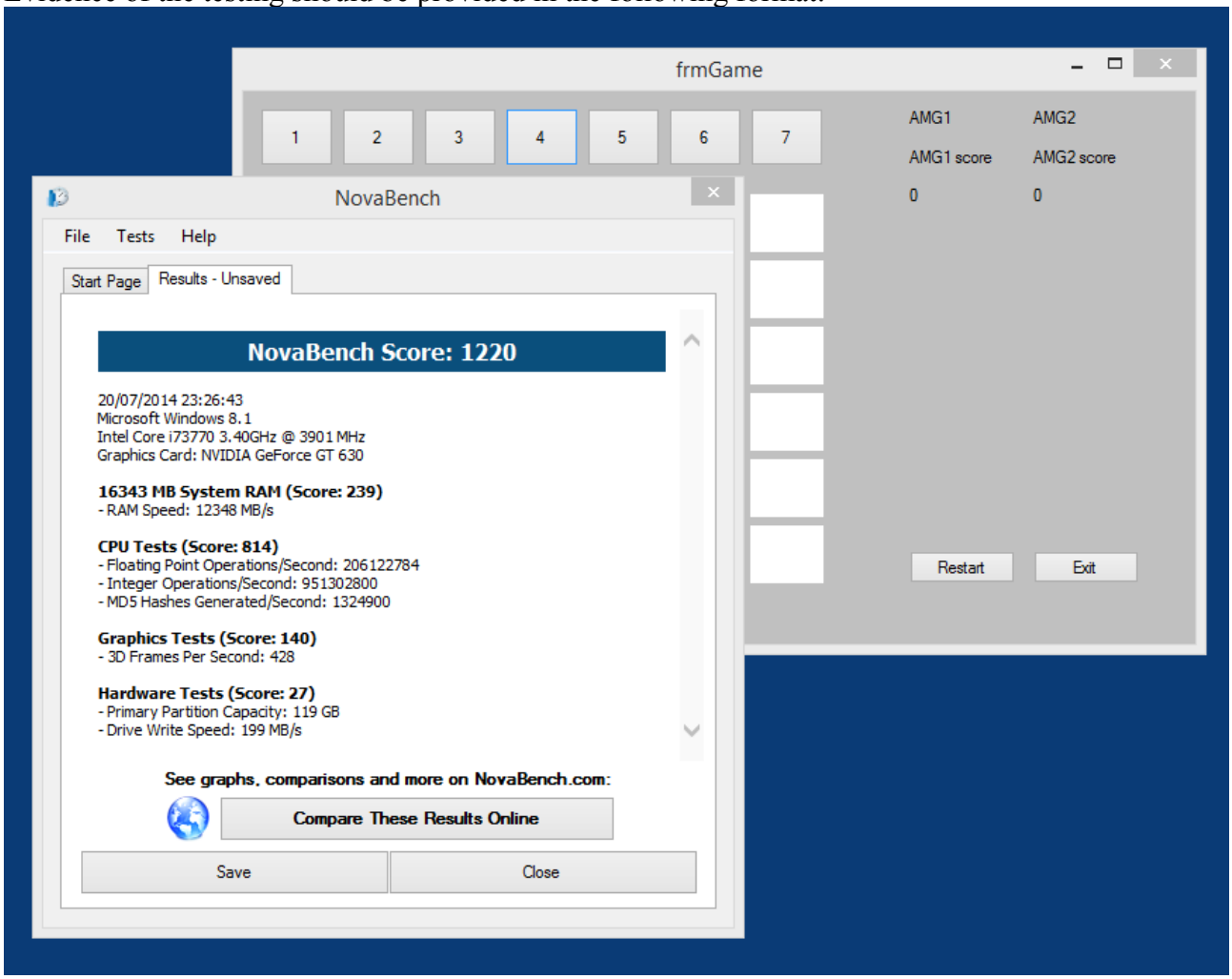
Key points to remember:

- Test data must check boundary conditions **and** either side of boundary values.
- If testing a subroutine which takes user input, for example a subroutine which validates Player Names, you should also check all types of input the user may provide. If the subroutine doesn't directly take user input, for example a function which will check whether a chess move is valid, you don't need to check for unexpected inputs such as strings.
- Your driver subroutine must be a *single subroutine* which passes all test data to the nominated subroutine. i.e. don't do one driver per test data set.

  Test data must be hard-coded into the driver. Do not obtain the test data from the user via a textbox, inputbox etc.

Testing on different computers

Evidence of the testing should be provided in the following format:



Note:

- NovaBench lists the computer specifications and provides benchmark data. https://www.piriform.com/speccy is an alternative product.
- The program is seen in the same screenshot.
- Some overlap (as above) is recommended.
- The text in the NovaBench screenshot must be clearly legible. You don't need to screenshot the entire screen. A partial screenshot like the one above (where all the text is legible) is expected.
- You must also provide a statement of how the program performed on the tested system. For example, did it run normally? Were there any delays?

Appendix E: Useful VB.net code snippets

None of these code snippets can be used to meet the minimum project specs requirements in relation to user developed subs and functions.

File references using relative paths

Throughout these code snippets you will see reference to functions such as `application.startuppath`. This is used so that your code will not rely on specific file paths, for example to your thumb drive or to your network drive. To determine where `application.startuppath` points to, you could create a button and attach this code:

```
MsgBox(application.startuppath)
```

It points to the \bin\Debug folder in your project's directory, but does not include the trailing backslash. If you receive "file not found" errors, inspect the error message for clues on how to resolve it. Be very careful of missing or duplicate backslashes, or duplicate file extensions.

Hiding and showing forms

`Me.Hide()` or alternatively you can use `Me.Close()`

```
frmMain.Show()
```

There is a very good explanation of using forms and the difference between hide and close in the book *Visual Basic 2005 Express: Now Playing*. Go to http://tinyurl.com/623mqn . Under the heading *Contents* select *more* >> then select *Designing the Windows of a User Interface* (p113). The most relevant section is at the bottom of page 114.

Launching an External File (e.g. pdf or ppt)

```
System.Diagnostics.Process.Start(Application.StartupPath &
"\Helpfile\Helpshow.ppsx")
```

If you do wish to launch a powerpoint file (e.g. for a tutorial) pps or ppsx is preferred to pptx or ppt.

Incorporating a help file

- Help files can be easily created using the freeware Helpmaker. Copies can be found in the peer drive or at http://www.vizacc.com/.
- To launch an external help file when a button is pressed, use this code:
  ```
  Process.Start(Application.StartupPath & "\helpfile\myhelpfile.chm")
  ```
- To link an external help file to the F1 button, use this code:
  ```
  Private Sub Form1_HelpRequested(ByVal sender As Object, ByVal hlpevent As
  System.Windows.Forms.HelpEventArgs) Handles Me.HelpRequested
  Process.Start(Application.StartupPath & "\helpfile\myhelpfile.chm")
  End Sub
  ```
- This assumes that you have a sub-folder called helpfile. Note that you need to put the _HelpRequested code on every form. It is possible to link different forms to different help files.

Reference: http://bytes.com/forum/thread505733.html (post 4). Post 5 contains an alternative method that you may wish to try.

Displaying images in a picturebox

```
Picturebox1.image = (Image.FromFile(Application.StartupPath & "\mypic.jpg"))
```

Randbetween function

```
Public Function RandBetween(Low As Integer, High As Integer) As Integer
  RandBetween = Int((High - Low + 1) * Rnd) + Low
End Function
```

Select last item in a listbox

This is useful for debugging or whenever you need to print a lot of lines to a listbox.

```
listbox1.SelectedIndex = listbox1.Items.Count - 1
```

Reading and writing sequential files

See the 9.2.2 notes.
The key commands are
`FileSystem.FileOpen(1, filepath, OpenMode.Input)` – use `OpenMode.Output` for writing
`FileSystem.Input(1, variablename)` – use `FileSystem.WriteLine` for writing to a file.
`FileSystem.FileClose(1)`

High Scores suggested method

See the 9.2.2 notes

Splitting words into individual characters

Strings can be read as arrays in Visual Basic, for example if myString = "hello" then myString(0) = "h". Note it is indexed from zero.

A loop can be written to assign each letter in a string to an element in an array.

A more complex method is:

```
Private Sub ExtractText(InitialText as String)
    Dim counter As Integer
    upper = Strings.Len(InitialText)
    For counter = 1 To upper
      letters(counter) = Strings.Left(InitialText, 1)
      InitialText = Strings.Right(InitialText, Strings.Len(InitialText) - 1)
    Next counter
End Sub
```

The mid function can also be used in conjunction with a loop.

Capture the index of a clicked control in a control array

`Dim cName As String = Sender.`**`Name`**
Note that this will capture the name of the control. Use `msgbox(cName)` to verify this.
You may need to use the `right` and `len` functions to extract the actual index.
Another approach is to use the `ValidateInteger` routine that was examined in the Preliminary Course (8.2.2). It will strip any letters from the name, leaving only the numbers.

```
Public Function ValidateInteger(ByVal InputString As String, ByVal Min As Integer, ByVal Max As Integer, ByVal DefaultVal As Integer)
As Integer
        Dim StringInteger As String, TempChar As String
        Dim Counter As Integer, IsADigit As Boolean

        StringInteger = ""

        'Consider each character in turn
        'If it is a digit then add it to StringInteger
        For Counter = 1 To Len(InputString)
            TempChar = Mid$(InputString, Counter, 1)
            'The ASCII code for the 10 digits range from 48 to 57
            IsADigit = Asc(TempChar) > 47 And Asc(TempChar) < 58
            If IsADigit Then
                StringInteger = StringInteger & TempChar
            End If
        Next Counter
        ValidateInteger = Val(StringInteger)

        'Check for at least some digits and that number is within the required range
        If StringInteger = "" Or ValidateInteger < Min Or ValidateInteger > Max Then
            ValidateInteger = DefaultVal
            MsgBox("A number from " & Min & " to " & Max & " is required.")
        End If

End Function
```

Note that the line above where cName is set, the Name property is retrieved (in bold). Depending on your control array, it may be better to return the .Text property. An example of this is in Hangman where you may have 26 buttons or labels containing the letters of the alphabet. If you return the text property (e.g. "A") then you simply subtract 64 from the ascii code of the letter to get the index (assuming it is indexed from 1)

Pausing/Sleeping Command

`system.threading.thread.sleep(x)` where x is the time in milliseconds.
Controls and forms can be updated by using the .update command. Examples of its use:

- `picturebox1.update`
- `me.update` (to update the form)

It may be necessary to use the .update command in conjunction with the sleep command, for example if moving a piece on a form.

Region label

For more complex projects, you can use the **Region** label to group and hide related sections, for example declarations.

Code is typed as follows:

```
#Region "Control Arrays"
    Private arrpicBoardTiles(0 To 15, 0 To 15) As PictureBox
    Private arrpicPlayerTiles(0 To 7) As PictureBox
    Private arrlblScores(0 To 4, 0 To 1) As Label
    Private arrlblPlayerOrder(0 To 4, 0 To 1) As Label
#End Region
```

It can then be collapsed and will appear as follows:

```
Control Arrays
```

Capturing key presses

This section was provided by Jeremy Ellingham from the class of 2012..

To set a key press/depress action trigger

1. Add into the Form Load or other Private Sub:
   `KeyPreview = True`

2. Click inside the code for any subroutine and navigate to the Events list top right (the lightning bolt symbol)

3. Choose KeyUp (for key release) or KeyDown (for key depress)

4. Inside the newly created subroutine, create a condition for the key that you wish to assign using e.KeyData = Keys.[your key]

For Example

```
Private Sub frmMain_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load

        KeyPreview = True

End Sub

Private Sub frmMain_KeyUp(ByVal sender As Object, ByVal e As
System.Windows.Forms.KeyEventArgs) Handles Me.KeyDown
        If e.KeyData = Keys.K Then
            MsgBox("you pressed the letter k")
        End If
End Sub
```

Dynamic object creation

This section was written by Andrew M Hall from the Class of 2014.

Many programs require a grid of objects such as picture boxes, buttons or labels. While this can be done by hand, it is often far easier and cleaner to write some code to generate the required objects dynamically within the program. The below method below can be implemented in a few minutes and eliminates the errors associated with manual object creation.

The theory behind dynamic object creation is that one can define an object within a program by declaring the object and giving it certain properties such as location, name and size. Since we are dealing with grids of objects, the location and size tend to be easily coded for. This method also allows objects to be easily linked to a common subroutine that will run when any of the objects are clicked.

```
Public Class Dynamic
    Dim object_array As Object          'This array will hold the objects
    'These objects must be declared globally
    Dim obj1, obj2 ... objn As New Object
    Private Sub Dynamic_Load(sender As Object, e As EventArgs) Handles MyBase.Load
        fill_array()        'Fill the array for manipulation
        create_objects()    'Define the objects
    End Sub

    Private Sub DoSomething_Click(sender As Object, e As EventArgs)
        'The variable sender will be set to eh object that activated the sub by clicking
        'Type code here
    End Sub

    Private Sub fill_array()
        'This sub will place all the required objects into an array for manipulation
        'This sub should be generated in excel
        object_array(1, 1) = obj1
        object_array(1, 2) = obj2
        .
        .
        .
        object_array(Width, Height) = obj & (Width * Height)
    End Sub

    Private Sub create_objects()
'This sub will define the properties of the objects that you want to dynamically create
        Dim x_coordinate, y_coordinate As Integer
'These are used to define the location of each object in pixels
        For y = 1 To Height_of_board
'This iterates through the board top to bottom
            For x = 1 To Width_of_board
'This iterates through the board left to right
                x_coordinate = (x - 1) * Width_of_object + initial_x_coordinate
'This formula will separate the objects by a set amount
                y_coordinate = (y - 1) * Height_of_object + initial_y_coordinate
'starting from an initial position
                With object_array(x, y)
'The With statement means that we can define the properties of the object
                    .Name = "type name here"
'The name property can be defined relative to x and y at will
                    .Location = New System.Drawing.Point(x_coordinate, y_coordinate)
'The location of each object will be in a grid fashion
                    .Size = New System.Drawing.Point(width_of_object, Height_of_object)
'The width and height of the object must be constant
                    .Font = New System.Drawing.Font("Standard", 12, FontStyle.Regular)
'Most objects have large numbers of properties that can be defined here such as
'Backcolour, Image, borderstyle, SizeMode and enabled
                End With
                Me.Controls.Add(object_array(x, y))
'We must put the object into the controls of the form, this makes it visible
'We can make a sub run when we click the object by using the addhandler
'We create a sub and can link multiple objects to it by linking them with an addhandler
                AddHandler object_array(x, y).click, AddressOf DoSomething_Click
            Next x
        Next y
    End Sub
End Class
```

Appendix F: CASEWHERE, ELSE IF advice

The following blocks of code perform the same function:

IF colour = “red” THEN
  message = “stop”
ELSEIF colour = “amber” THEN
  message = “stop if possible”
ELSEIF colour = “green” THEN
  message = “proceed”
ELSE
  message = “use caution”
END IF
	CASEWHERE colour
	  “red” : message = “stop”
	  “amber” : message = “stop if possible”
	  “green” : message = “proceed”
	OTHERWISE: “use caution”
	ENDCASE

For simple comparisons, the CASEWHERE statement provides easier-to-read code and is preferred.

Note that the NESA SDD Course Specifications document doesn’t recognise ELSEIF structures.

For your projects, you are advised:

- If a single variable/value is being tested, and there are only 2 possible outcomes, use IF/ELSE/END IF.
- If a single variable/value is being tested, and there are 3 or more possible outcomes, use CASEWHERE.
- If the condition being tested involves more than one variable/value joined with AND/OR, use IF/ELSE/ELSEIF as needed.

In Visual Basic, the CASEWHERE control structure is implemented as follows. Note that a range of values can be easily tested.

```
Dim number As Integer
number = 8
Select Case number
    Case 0 To 5
        Message = "Between 0 and 5, inclusive"
        ' The following is the only Case clause that evaluates to True.
    Case 6, 7, 8
        Message = "Between 6 and 8, inclusive"
    Case 9 To 10
        Message = "Equal to 9 or 10"
    Case Is < 0
        Message = "Negative number"
    Case Else
        Message = "Not less than or equal to 10"
End Select
```

References:

- http://www.dreamincode.net/forums/topic/180080-ifelse-vs-select-case-comparison/page__view__findpost__p__1458479?s=b5bf88b2dffaed3dff6bbc7c9a6c1478 – see comment 12 for some of the principles involved.
- http://msdn.microsoft.com/en-us/library/cy37t14y.aspx for the Visual Basic Select Case format.

Appendix G: Syllabus Content

This syllabus extract will give you an indication of some of the syllabus areas being covered by the project.

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

Students learn about:	Students learn to:
• implementing – converting the solution into code – systematic removal of errors – refining the data dictionary – including standard or common modules or subroutines – using software to refine documentation – creating online help – reporting on the status of the system at regular intervals – applying project management techniques • testing and evaluating – completing thorough program and system testing – completing all user documentation for the project • maintaining – modifying the project to ensure: - an improved, more elegant solution - all needs have been met - the software solution operates under changed environments or requirements – updating the software specifications and documentation to reflect the changes	• 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 • ensure that relevant ethical and social issues are addressed appropriately • evaluate the project in relation to the original understanding of the problem • review and evaluate the quality of the solution making the necessary changes
Whole project issues • project management techniques • social and ethical issues • feedback from users at regular intervals	• manage the project effectively • communicate effectively with potential users