SOFT050 Computing Project

Assignment Title:	Computing Project
Submissions:	1. Proposal submission
	Deadline: Fri 23 Jan, 5pm
	Submitted at: Faculty office
	Format: Paper copy
	2. Final submission
	Deadline: Fri 24 Apr, 5pm
	Submitted at: Online (DLE)
	Format: Electronic zip archive
Demonstration:	Time TBA
Contribution to Final Grade:	100%
Individual/group assignment	Individual assignment
Module Staff:	Dr Torbjørn Dahl (module leader), Dr Liz Stuart

Module Aims

1. To enable students to apply the skills, concepts and experience acquired in other modules of the programme to a project in the students' chosen discipline.
2. To present the work done in this module in both written format and orally.

Learning Outcomes

At the end of the module the learner will be expected to be able to:

1. undertake the planning and execution of a project, identifying the design requirements and evaluating the work done at various stages of the project;
2. write a technical report on the project and present the work orally.

1 Specification

This is an open-ended project where it is up to you to define the specific content within the guidelines given below. It is an opportunity for you to develop an application of significant scope and complexity drawing on the skills you have acquired during your studies. Possible projects may include desktop applications, smartphone apps, dynamic web sites or embedded controllers. The main criterion for an acceptable project idea is that it requires you to independently develop a significant amount of program code.

2 The Proposal

The purpose of project proposal is to ensure that your chosen project is appropriate in terms of content and scope. It assesses the scope of your proposed project and enables us to guide you, as appropriate. This prevents you from spending a lot of time on something that will not get you a good mark.

A proposal template has been provided on the module Moodle site. All the elements in the template are required.

2.1 Deliverables

This section of your project proposal should list what it is that you will be producing. This list should include all the tasks, related to your product, that you need to complete as well specifying milestones to be reached. Note you are required to have at least 4 technical deliverables.

Technical **deliverables** are specific to your own project. They reflect a significant part or feature of your final product. Examples include: design of the graphical user interface, WAMP server installation or implementation of collision detection in a game.

To make it easier to manage your time, you should describe your deliverables using nouns. It is also recommended that you make each deliverable SMART: Specific, Measurable, Attainable, Relevant and Time-bound. This will make it is easier to know when a deliverable is finished. An example of a bad deliverable is "research available development tools". A good example is "A running database containing ten or more example animals".

Each deliverable should be sub-divided into a set of **tasks** that must be completed in order to produce the deliverable. The completion of a deliverable or task may define a **milestone.** Milestones are used to mark the end of significant project stages, e.g., the completion of infrastructure prototypes or the completion of the user testing. Milestones are used to check that a project is **on-time**, but a milestone is also a **decision point**. When you reach a milestone or the date of a milestone, you need to make a decision.

Examples of Milestones

- Let us suppose that you have completed an evaluation of different server-side scripting languages. At this stage, you will need to decide which language to use.
- Let us suppose that you have managed (or failed) to display a video using the wxWidgets library. At this stage you will need to decide whether to (i) try another GUI library or (ii) drop the video-based features.
- Let us suppose that you have completed user testing. At this stage you need to decide what changes to make as a result of the feedback you have received.

It is up to you to decide when you want to set your milestones and what tasks and/or deliverables (if any) they should be related to.

Finally, you should add priorities to your deliverables, e.g., using the MoSCoW (Must, Should, Could, Won't) framework[1]. This will help you schedule the features by focusing on the most important features first. It will also help you to discard features if you run shot on time.

2.2 Schedule

This section of your project proposal should list all of the tasks needed to produce the deliverables. In addition, it should also specify timings for each task and any project milestones.

To produce a schedule, each task should have an estimated **duration**, i.e., you should have a guess at how long it will take you to complete it. Each task should also have an allocated **time-period** between the start of the project and the final submission deadline. Note that tasks that take longer than one week should be broken down into multiple sub-tasks.

A schedule is needed for two reasons. Firstly, it enables you to verify that you have enough time available to complete the planned work. If you find that you cannot complete the deliverables within your available time, you must reassess the list of deliverables. Most likely, you will need to drop some deliverables or make more time available. Most importantly, be realistic when you schedule tasks.

> *"In preparing for battle I have always found that plans are useless, but planning is indispensable."*
>
> Dwight D. Eisenhower

The second reason for needing a schedule is that your estimated task durations are likely to be wrong. For example: it is likely that you will find that you need more time than you thought to complete a particular feature.

In addition, external factors (such as illness) will change the amount of time you have available. You may start missing milestones. When these things happen (and regrettably they do regularly), you should **immediately reassess your schedule**. It is likely that you will still be able to produce an acceptable end result by changing your schedule accordingly. This may mean dropping some deliverables (reducing the scope of the project) or it may be possible to manage the situation by making more time available.

Please note that is it not a good idea to get your schedule back under control by simply reducing your estimates of task durations to "make them fit" into the time you have left. You need to be proactive and strategic. You need to scope your project again to ensure you have a suitable deliverable to submit at the end. Without a realistic schedule you are likely to be left with too little time to produce a project of the quality you initially wanted.

2.3 Risk Analysis

*This section of your project proposal (refer to section F in **Error! Reference source not found.**) lists the resources required for development, evaluation and presentation and an evaluation of the availability of these resources.*

Your risk analysis should contain a list of things that can go wrong with your project. As a minimum, this should include plans for what to do if any piece of hardware stops working. In particular, you

[1] The MoSCoW framework is described in further detail on Wikipedia.

should consider how to back up your code and your final report. A free repository such as Bitbucket[2] is a good option.

It is also a good idea to think about how you will change your project if you miss any major milestones, i.e., prioritize the deliverables so that you know which one to drop first.

You may also include other things that can go wrong, in particular things specific to your project.

3 Final Submission

The final submission consists of two parts, the source code and the final report. Both parts contribute to your grade by demonstrating your knowledge and skills as detailed in Section **Error! Reference source not found.**.

3.1 Source Code

This should contain all the code you have collected and/or written for your project. Your own contribution should reflect around 50 hours of independent work.

- As far as possible, the submitted code should contain copies of the folders and files needed to run/view your project.
- Remember to include any supporting software libraries you have developed or used. However, if you rely on a major framework such as the Apache web-server, it is clearly not feasible to include everything.
- The report should state clearly what code you have taken from other sources, e.g., code libraries, and what code you have developed yourself.

3.2 Report

The report should be a Microsoft Word document of no more than 2,000 words. Please use screen shots and diagrams to illustrate the design and functionality of your product. A suggested report structure is provided in Table 1. Note that the code should not be a part of the written report, it must be submitted separately.

Section	Content
Introduction	• Summary: Explain briefly what the project is about • Motivation: why you chose this project over other alternatives
Technology review	• Present the technologies you used and possible alternatives • Explain why you chose the technologies you used
Design	• Your design for the project, e.g., interface sketches, user stories, UML class and deployment diagrams
Implementation	• Effort: explain what you did to realise the project • Results: explain what you produced, include diagrams and screenshots when possible
Evaluation	• Procedure: how did you evaluate your implementation, e.g., test cases, user studies? • Results: What were the results of your evaluation? What aspects of

[2] https://bitbucket.org/

	your application worked as well as, or better than, expected? What could be improved and how?
Reflection	• Project Management: What changes did you have to make to the original proposal, including the schedule? • Experiences: What went well? What didn't go so well? • Lessons: What you learned from the project? • Recommendations: What will you do differently in your next project?

Table 1: Suggested Report Sections

4 Demonstration

The presentation will be made to a panel of staff members, including at least one member of the module staff. Your demo will last around 15 minutes. It begins with you demonstrating and explaining your project. After a short demonstration, the panel will question you on the functionality and details of the implementation. Questions will test your understanding of the project workings and its code. This also helps us verify that you have written the code yourself. Thus, you are strongly advised to read through your code, prior to the demonstration, to ensure you are familiar with it when examined.

You are responsible for organising the facilities necessary to demonstrate your project. In particular you must ensure that any software and hardware required is available. This may mean that need to bringing them yourself or that you need to book them through the appropriate University channels.

It is critical that you bring your source code to the presentation as the panel will ask you to show them and explain the code behind selected features of your project. A failure to do this places you in significant risk of failing the demo.

5 Submission

5.1 Proposal

The proposal must use the provided template. Provide as much detail as you can in the proposal. This will help you understand and plan your project. IT will also allow the module team to provide better feedback.

5.2 Final submission

The final submission must be a single zip archive file uploaded using the online submission mechanism on the module Moodle site. The archive must contain both your source code folders and files and the written report.

Failing to submit your source code, e.g., by submitting only the project structure files or compiled files, will result in a fail grade.

According to university regulations, late submissions up to 24 hours past the submission deadline will be accepted, but their marks will be capped at the lowest pass mark (40%). If you have problems completing your project, please look at the university's policy on extenuating circumstances.

The work submitted must be developed by you independently. Submitting code that has been produced by others (or in collusion with others) as if it was your own is considered unfair academic practise and may have severe consequences including cancellation of grades and expulsion from the university. You may use freely available code and/or content as long as you explicitly list these in your report and including the licence under which they are available.

6 Marking

The project contains three assessed elements which contribute differently to the final module grade. The details of the individual assessment elements and their weightings are given in Table 2.

All the elements of the assessment (including the proposal) must be passed. Failure to submit a proposal or to attend the demo will result in an overall fail grade for the module.

Assessment elements	Evaluated Through	Weight
1. Technology review	Report	10%
2. Design and usability	Demo and report	40%
3. Code quality and understanding	Demo, code and report	40%
4. Project execution and reflection	Proposal, demo and report	10%

Table 2: Assessment element weights

A rough indication of what is required for each grade level is given in

Grade	Typical Project
1st (>70%)	The application runs, is robust to unpredictable users, is a professional looking and easy to use, it implements a rich and well-rounded set of features and the code is well structured and includes high-level structures such as functions, classes and objects in multiple files.
2.1 (60%-69%)	The project runs and presents a wide range of interactive features, that are suitable for the given application, the application is easy to use, there is a lot (100s of lines) of well-structured code and the programming logic
2.2 (50%-59%)	The project works and contains a good amount of functionality and user interaction and there is a significant amount of well-structured code using rich logic and data structures.
Pass (40%-49%)	The application is simple but runs. The program logic implemented is limited, but includes the basic programming elements (variables, if-statements and loops), there is little or no user interaction, the code is clumsy, e.g., with many repetitions instead of loops.
Fail (<40%)	The application does not run, the student has little or no understanding of the code or the application contains only a small amount (<50 lines) of code developed by the student

Table 3: Typical properties of project of different quality.