CSC3414
CSC3415

Software Engineering Project A & B

Faculty of Sciences

Introductory/Study Book
2003

Written by

Dr Richard Watson
Department of Mathematics and Computing
The University of Southern Queensland

This book was set in Computer Modern Roman and Helvetica by the author, using the $\LaTeX$ typesetting system and a set of special macro definitions. It was produced on a IBM compatible PC running under RedHat Linux.

Published by

Distance Education Centre
The University of Southern Queensland
Toowoomba Qld 4350
Australia

http://www.usq.edu.au/

Contents

Course Specification

Insert course specification here

Welcome to the Software Engineering Project courses. These two courses are intended to be taken in the final year of your program, and are mandatory courses within the Software Engineering major of the Bachelor of Information Technology.

These courses are unlike most other courses that you have undertaken at the university. There is no specific textbook and/or study book to provide technical infromation and there is no examination.

Instead you have the opportunity, and the challenge, to apply knowledge that you have acquired throughout your program in order to complete a large software project.

The project is completed over two semesters. Course CSC3414 is undertaken in Semester 1, and is comprised mostly of analysis and design activities. The Semester 2 course, CSC3415, focuses on implementation and testing.

These courses have many aims. The major ones are:

- To allow you to demonstrate that you are able to apply knowledge gained from previous courses.
- To provide you with the experience of working on a software project of significant size. You will be writing thousands of lines of code. Obviously this will require a high level of planning on you part. It also prepares you for some of the kind of work that you will encounter in the IT industry after you graduate.
- To give you practice in creating design and other project documents.

Details of the procedures and requirements follow. The most important ingredient for success in these courses is your own motivation. Right from the beginning you must work consistently on this project. Planning is vital. Continual progress (even if it seems small) is vital. If you fail to adopt a professional and consistent approach to completing these courses you will find it very difficult to pass. You simply cannot leave important work until the last few weeks of the semester.

You should adopt a professional approach to your work. This is not “just another assignment” which you must submit to get your degree. It is a practice run for your future career. So it behoves you to work like a professional, that is:

- Be well organised.
- Produce high quality work that responds to the (imaginary) client’s needs.
- Work consistently and deliver on time.

Please don’t read these warnings in a negative light; these courses may well be some of the most rewarding that you have undertaken during your program. When approached with an enthusiastic and confident attitude, these courses can be as exciting and satisfying as they are important for your development as a software engineer.

1 Course Organisation

1.1 The project

In these courses you are provided with a fairly detailed set of requirements for a software system. Your job is to analyse these requirements, then design, implement and test a system to meet these requirements. Along the way you must demonstrate skills in planning the project and tracking its progress, in creating various analysis and design documents, and in documenting the user interface to your system.

It is vitally important you realise that, from an assessment perspective, we are interested just as much in *how you developed the system* as we are in the final software product. See section 1.7 for details of what you must do and what assessment weight is carried by each item.

The system is based on a **User Requirements** document, which appears as section 3 of this book.

1.2 Role of the Course Examiner

The course examiner has three basic roles.

1. To act as the 'client'. Imagine that he or she is paying you to do this work. You want to do a good job and produce what the client really wants. So if you have any uncertainty about what you should do based on the (perhaps limited) user requirements document you must contact the examiner to clarify the position.
2. To act as a mentor. If you are unsure how to proceed, or perhaps have some technical questions, do not hesitate to contact the examiner.
3. To assess the completed work.

The examiner will be available for consultation either by telephone, email or face to face at his or her office.

Note that in some cases, the course examiner may nominate someone else to assume these roles and so become your project supervisor. Nonetheless, the examiner has final responsibility in awarding grades for students of the course.

1.3 Role of the Student

Obviously your major role is to finish the job on time, completing all assessable items satisfactorily, and within budget!

However, in order to do this you may well have to undertake some private study of your own in order to learn new skills which you will need to apply in the development of the project. You may also need to re-learn topics that were covered earlier in your studies, or perhaps study some of these topics in more depth than was done originally.

You cannot assume that, at the beginning of these courses, you have all the knowledge and skills necessary to complete it satisfactorily.

There are two other key requirements:

1. **You must communicate regularly** with the course examiner or the supervisor nominated by the examiner. Just as a software project manager expects and receives regular progress reports from software engineers working on a project, the course examiner expects to hear from you.

   Specifically, you **must** report on a weekly basis. This can be done either face to face (for on campus students) or via email (students in either mode). The examiner may arrange a weekly group meeting — attendance and verbal reporting at such a meeting would be sufficient.

   You will find that maintaining a diary (see section 1.6.3) will help you greatly in reporting weekly progress to the examiner. Reports can be brief, but they must be detailed. You must be able to report time spent on specific tasks, rather than general comments about your progress.

   Note that communication and reporting is assessable; 10% of each assignment's marks are assigned for this component.

2. **You must do your own work.** It is quite acceptable to discuss the design and implementation with other students, and to assist each other in overcoming technical problems, but the actual material that you submit for assessment must be your own work.

   It is completely unacceptable to copy even small parts of another student's work, or for a number of students to collaborate to produce an identical item.

   Unacknowledged copying is plagiarism. The University has strong regulations which provide substantial penalties for students shown to have plagiarised.

A final strong recommendation:

- **Do your best.** Aim to do a really good job. View this project as preparation for your future career. Rather than thinking "What is the minimum I need to do to pass?", ask instead "What can I do to show that I really know my job?".

1.4 Computing Platform

You may implement your project under ether Microsoft Windows (e.g. 95, 98, NT, 2000, XP) or Linux. Other commercial platforms are not suitable as we do not have access to such systems for testing purposes.

The examiner will need to be able to install and run your software. You cannot rely on the availability of proprietary software such as Visual Basic. (Note however that most applications like VB allow the user to build an executable version of the software which does not require the development environment.) You must ship a final implementation that can be run on a bare Microsoft Windows or Linux system.

If you use the Linux platform, you may use any standard application shipped with the Department's distribution of RedHat Linux.

1.5 Resources

It is assumed that most of the resources that you will need to complete this project will come from courses that you have undertaken in the past. As discussed in section 1.3, you may need to access other material to advance your skills and knowledge.

The bibliography at the end of this book lists a selection of possibly useful texts in the areas of software engineering, and systems analysis and design. You should also consult the USQ library.

1.6 Overview of assessment process

The full project extends over two semesters. The first semester concentrates on the tasks of system analysis, system design and project planning. No code should be written, except perhaps to prototype some features like user interface or to try out unfamiliar implementation techniques or tools.

The second semester is devoted to implementing the project. In addition to coding the system, a comprehensive test plan must be drawn up and a testing system implemented. User level documentation is created.

There are two assessment items due each semester. The exact details of each assessment item are described in section 1.7. The due dates appear in section 2.

1.6.1 Semester 1 tasks

1. Analyse the informal user requirements document provided to you and produce a precise and detailed definition of requirements.

2. Design the software system. This ranges from overall architecture to precise details such as definition of class templates, record or database structure, and user interface dialogues.

3. Plan the complete project. This involves identifying tasks, estimating how long they will take to complete, and determining when they will be undertaken. You will also record actual hours spent performing each task.

1.6.2 Semester 2 tasks

1. Develop and use a comprehensive test plan. The following suggests some test plan activities — it is neither comprehensive nor is it prescriptive. You may add or delete activities as fits your implementation environment. However you must have a clear and comprehensive plan.

   - document the test cases (inputs and expected outputs)
   - plan when to test
   - record test results
   - automate the test process to enable efficient repeated tests

2. Implement the system. This includes developing installation instructions, packaging the software for distribution and possibly providing an installation program or script.

3. Produce user documentation. This is a well structured document describing how to use the system. It is separate from installation instructions.

#	Task	Estimate (hours)	Actual hours per week				Total
			week 1	week 2	week 3	· · ·	
1	Blah blah	32	2	7	2	· · ·	11
2	Grumble	50	6	0	1	· · ·	7
3	Hurrumph	18	0	5	6	· · ·	11
	. . . .						
	Total	· · ·	8	12	9	· · ·	29

Table 1: Sample task list showing estimates and progress

1.6.3 Record keeping

Here is a very strong recommendation: **keep a log or diary** of time spent on this project. Each day you should log the hours spent against the task(s) you are working on. This can be done electronically, using a spreadsheet, simple text file, or special purpose project planning software. Also keep a paper copy in a standard diary — these entries are less likely to be lost, and it is very quick and easy to jot down a few lines in a diary, then later log them on the computer.

Keeping a diary is essential if you want to accurately keep track of time expended, and you are required to present these details in the reports.

You should maintain a table of tasks and effort expended against each task. The list of tasks is established early in the project and will most likely grow as the project progresses. You may wish to use a spreadsheet to record this data.

A typical format is depicted in Table 1. You are required to submit multiple versions of this task list, at various times during the project. Later versions differ from earlier ones by having more columns completed, and perhaps by having more rows as more tasks are identified. However, *do not change initial estimates.* You will not be assessed on how well your actual effort matched your estimated effort — rather you should use the figures to help learn about the difficult task of software cost estimating.

1.6.4 Evolving documents

You will be asked in section 1.7 to submit the same document on more than one occasion. For instance you must provide a task log four times. Also, you are asked to provide a number of versions of the analysis and requirements documents.

When submitting a revised document you must include either as an annotation inside the actual document, or as an accompanying commentary, some indication of what has changed since the previous version of the document was submitted. Changes to the task log are easy to describe (e.g. added new tasks..., new logs for weeks 7...12) but the other documents will require more effort. For requirements documents, one workable and widely used scheme is to include a section (or maybe an appendix) in the document which provides a log of changes made. For example the change log could be a simple chronological list which might in part look like:

2/4/02	added new requirements 3.4, 3.5
22/4/02	modified requirement 7.4 to permit supervisor as well as clerical staff to modify database.
5/5/02	...

1.6.5 Document conventions

You will produce many documents as part of the assessment process. These should be well organised, readable and well formatted. Hand written documents are not acceptable. All documents should contain the following:

- A title; larger documents should have a separate title page.
- The name of the author.
- The date that the document was produced.
- The version of the document.
- Numbered sections. The use of numbers allows cross referencing within and between documents (a characteristic of good documentation) and makes it easy for the reader to understand the hierarchical nature of the document.

1.7 Assessment requirements

Two “assignments” are due each semester. Really they would be better described as “project deliverables”. Some of the deliverable items are “work in progress” and updated versions are submitted at a later date. Make sure you read section 1.6.4 for the correct way of handling the creation and maintenance of these evolving documents.

The weighting attributed to each assignment is a per-course contribution. For example, the first assignment for CSC3414 contributes 50% of the marks for course CSC3414.

CSC3414	Assignment 1	weighting 50%

Submit the following items. They should be well formatted and carefully prepared documents. Hand written documents are not acceptable. See section 1.6.5 for more details about document format.

1. **User requirement analysis** **weight 10%**
   Read and analyse the user requirements presented in section 3 and produce a report on your findings. This should be written in natural language but well organised and structured. The report should do *at least* the following:

   - Identify ambiguities and inconsistencies in the report.
   - Identify missing or incomplete information.
   - Describe how you intend to handle these problems. Typically you will either need to
     - make assumptions (which must be fully documented) or
     - seek clarifications from the "user" (which must also be fully documented).

   Include any other comments that you think are important in this report.

   References: See [8] (chapter 4) or [9] (chapter 5) for more on analysing user requirements.

2. **System requirements document** **weight 30%**

   This is a more detailed and complete version of the User Requirements document. The extra detail comes to a large extent as a result of the analysis phase described in the previous item. For instance, an assumption or clarification documented as such in the analysis phase will become a requirement in this current document. Where appropriate, each requirement should be supported by a justification and possibly a cross reference to the requirements analysis document or the user requirements document.

   The system requirements document sets out what you plan to deliver. *It does not describe design or implementation details.*

   The major element of the document is a complete, enumerated, well structured, and detailed list of requirements.

   If necessary, the document may be supported by models showing for example

   - relationships between elements of the user domain,
   - flow of information in the system,
   - state transitions, and
   - timing of events.

   References: See [8] (chapters 4 and 7) or [9] (chapter 5) for more on defining systems requirements.

3. **Preliminary plan and activity log** **weight 5%**

- Develop a preliminary list of tasks required to complete the project. This includes tasks undertaken in both of the semesters.
- Estimate the amount of time required to undertake each task. You should use hours as a measure of duration, bearing in mind that you should be spending about 10 hours/week on these courses.
- Record the time (in hours) spent undertaking each activity.
- Present the results in a form similar to that shown in table 1. Note that you may wish to break major tasks into subtasks.

4. **Communication** **weight 5%**

   As described in section 1.3, communication with the examiner or nominated supervisor is vitally important. You will be assessed on the regularity, timeliness, accuracy and relevance of your interactions.

CSC3414 Assignment 2 weighting 50%

Submit the following items. They should be well formatted and carefully prepared documents. Hand written documents are not acceptable. See section 1.6.5 for more details about document format.

1. **Updated User requirements documents** **weight 5%**

   This is an updated version of the documents 1 (User requirements analysis) and 2 (System requirements) produced in Assignment 1 earlier in the semester. Make sure that the changes to these documents are clearly described as discussed in section 1.6.4.

2. **Software Design Document** **weight 35%**

   Produce a document which describes *in great detail* what you are going to produce. You should be able to hand *only* this document to another programmer who knows nothing of the user requirements and expect the correct product to be produced.

   The following is a suggested list of some of the items that should be contained in the document. You will almost certainly need more than these. You will almost certainly not need to include some of these.

   - System architecture.
   - Detail data structure design. Examples are file or database formats.
   - Class templates in the target language.
   - Structure charts.
   - Data flow diagrams.
   - Pseudo code.
   - State transition diagrams.
   - User interface specifications.
     - How many screens? What are the input fields on each screen? What validation checks are made on fields? What are the initial values for fields?
     - If command line, what is the syntax for commands?

3. **Updated plan and activity log** **weight 5%**

   This is the latest version of the same item submitted in the first assignment. It should reflect *all* the work done during the semester.

4. **Communication** **weight 5%**

   As described in section 1.3, communication with the examiner or nominated supervisor is vitally important. You will be assessed on the regularity, timeliness, accuracy and relevance of your interactions.

CSC3415 Assignment 1 weighting 40%

In the course CSC3415 you will implement and test the system designed in course CSC3414. The first assessment focuses on testing. Of course you will be occupied mostly with coding, but it is imperative that you begin testing early. Hence we demand that you produce a comprehensive test plan. You should also automate the testing process as much as possible. This may mean writing extra scripts or programs which exercise your code and record the results. This will enable you to regularly test your code. (Note that automated testing with GUI application development systems like Visual Basic is difficult, and will not be expected.)

Submit the following items. They should be well formatted and carefully prepared documents. Hand written documents are not acceptable. See section 1.6.5 for more details about document format.

1. **Updated requirements documentation** **weight 5%**

   During implementation you will no doubt discover that changes are required to your original design. Edit the following documents, and include a change log, to reflect these changes.
   - User requirements analysis
   - System requirements
   - Software design

2. **Test plan** **weight 20%**

   Describe your approach to testing. For example (this is not an exhaustive list):
   - How do you test?
   - When do you test?
   - Have you created special test programs and scripts?
   - How do you record test results?

   You must document the test cases; each test case describes the required result from a given set of inputs and a specified system state.

   See [8] (chapters 22 and 23) or [9] (chapters 19 and 20) for more on testing.

3. **Version 0 system** **weight 5%**

   By this time you should have some executable code. It may be just the top level with some stubs, or some bottom level elements with some test driver programs.

   You should submit this code together with installation instructions. We require this for a number of reasons:
   - To gauge that you are making progress.
   - To ensure that the system will in fact install on our software platform.
   - To ensure that you think about the installation process now, rather than doing it in a rushed (and possibly inadequate) manner at the end of the project.

4. **Updated plan and activity log** **weight 5%**

   This is the latest version of the same item submitted in the previous semester. It should reflect *all* the work done during last semester and this semester.

5. **Communication** **weight 5%**

   As described in section 1.3, communication with the examiner or nominated supervisor is vitally important. You will be assessed on the regularity, timeliness, accuracy and relevance of your interactions.

CSC3415 Assignment 2 weighting 60%

This is the final submission. Make especially sure that everything is nicely packaged, well organised and neatly formatted. See section 1.6.5 for more details about document format. This is what is required:

1. **Final version of requirements documentation** **weight 3%**

   Submit the following documents.
   - User requirements analysis
   - System requirements
   - Software design

2. **Final version of plan and activity log** **weight 4%**

   This should reflect *all* the work done during both semesters of the project.

3. **User documentation** **weight 10%**

   This is an instruction manual for the user of your system. It does not contain installation instructions. Assume the user knows nothing about the technical details of the system. You can assume that the user has reasonable domain knowledge, and will be familiar with the kinds of 'things' with which your system deals.

4. **Test log & updated test plan** **weight 8%**

   Resubmit the final version of your test plan from the previous assignment. Most importantly, include a log showing when testing was carried out and the results of the testing.

5. **The completed executable system** **weight 30%**

   The executable programs that make up the system must be submitted, as well as the source code. You must include instructions to install the system.

   The following aspects of system will be assessed:
   - The distribution should be properly packaged.
   - The installation process should be well documented, simple, and error free.
   - All functionality defined in the User Requirements document should be met.
   - The system should be free of defects — that is it should not crash or behave incorrectly.
   - It should be usable — the user interface should be simple and natural to use.

6. **Communication** **weight 5%**

   As described in section 1.3, communication with the examiner or nominated supervisor is vitally important. You will be assessed on the regularity, timeliness, accuracy and relevance of your interactions.

1.8 Submission of project components

Hard copies of reports, suitably bound, should be submitted in the usual manner.

Source and executable code, and online documentation, should be submitted on floppy disk or CDROM. You should always package all files into a single archive file using `tar` and `gzip` for Linux or `zip` for Microsoft Windows environments. At least one backup copy of the archive should be provided if submitted on floppy. A file named `README` should appear on the media with comments about what is in the distribution file and how to unpack it.

If submitting executable code, you must provide instructions and possibly automated procedures for installing the application(s). These may appear in the `README` file or be included in a separate file name `INSTALL`.

1.9 Residential School

There is no residential school for these courses. If however you will be in Toowoomba attending a residential school, feel free to contact the course examiner if you wish to discuss any aspects of the course.

1.10 Communication

A number of avenues are available for you to communicate with the University and the course examiner.

- You can contact the course examiner directly via email at `lec.CSC3414@usq.edu.au` or `lec.CSC3415@usq.edu.au`. This is the preferred method for enquiries about academic aspects of the courses.
- You can also contact Outreach Services or the International Office, depending on your enrolment status. You can use this contact point for either administrative or academic enquiries. See below (under Student Enquiries) for details. Academic enquiries will be forwarded to the course examiner.

1.10.1 Linux Installation Problems

A systems programmer is employed in The Department of Mathematics and Computing to support staff and students in installing Linux. If you have problems installing Linux, first of all re-read the instructions and try again. If you still have problems contact Linux support via either of the following means:

- email to `linux@usq.edu.au`
- phone the Departmental secretary (07 4631 5555), or the DEC phone number listed in the following section, and ask to be put through to Linux support.

Please, PLEASE, PLEASE, before doing this investigate all options by reading the documentation, and make clear notes about your problem such as the precise steps you followed and any messages you received.

1.10.2 Student Enquiries

You should carefully read the information provided in your *USQ Student Guide* concerning contact details and support services.

Enquiries via the Internet

If you have Internet access, **USQ***Assist* is the most efficient method for requesting support assistance. This is a web self-service facility for **all** students. You can:

- find answers to common questions;
- ask a question; and/or
- track the progress of a question.

By typing a keyword in the search field, you can find answers to many of the questions frequently asked by students, including course troubleshooters. To access USQAssist, go to `http://usqassist.usq.edu.au` or click on 'USQ*Assist*' in USQ*Connect*.

Enquiries via Telephone or Facsimile

Alternatively, you can ask for support via telephone or fax.

	International Students	**All Other Students**
	Contact your Local Support Office for further assistance. If there is no Local Support Office in your country you should contact the International Office at USQ.	All administrative queries should be directed to the Outreach Services in the Distance Education Centre (DEC) or your Regional Liaison Officer.
Telephone:	61 7 46312362 (International Office)	07 46312285 (Outreach Services)
Fax:	61 7 46362211 (International Office) 61 7 46359225	07 46361049 (Outreach Services)
Web Form:	`http://usqconnect.usq.edu.au/usqassist`	
Email:	`iosupport@usq.edu.au`	`outreach@usq.edu.au`

2 Timetable

These courses should be undertaken over two consecutive semesters. As there is no set programme of study the only timetable elements are the due dates for the assignments. The assignment due dates are as follows:

Course	**Assignment**	**Date Due**
CSC3414	1	17 April 2003
	2	13 June 2003
CSC3415	1	5 September 2003
	2	31 October 2003

3 User Requirements

3.1 Overview

You are required to build a system called “PostMark” to record marks for students undertaking a university course. Typically assignment and examination marks are recorded but there may be (many) other categories such as, but not limited to, laboratory and field work, and assessment of creative art.

Furthermore, PostMark should calculate a student’s aggregate mark for the course based on all assessment items.

Using this aggregate mark, together with some rules, PostMark will determine a suggested grade to award a student. This grade can be over-ridden by the examiner.

3.2 The Players

PostMark deals with three kinds of users:

1. The examiner. There is usually only one of these, but more than one should be allowable. This user can do anything! This includes setting up configuration information, entering marks, and assigning grades.
2. A marker. A marker can only enter marks of assessment items that he/she has marked. There may be many markers. The system will keep track of who is marking which assessment item.
3. A student. A student can view her/his assessment record, but cannot see any other student’s record.

Because there are different levels of user, a password protected accessing scheme will have to be implemented.

3.3 Interface

The functions of the system should be presented to the user via a graphical user interface of some sort. There is a great deal of choice here depending on platform and level of sophistication desired.

Here are some implementation suggestions. They are by no means exhaustive, but merely presented to give some inspiration and provide a starting point for further thought.

- Tk; supported by either Tcl or Perl.
- Visual Basic.
- C++ with a GUI library.
- Java with an appropriate GUI library.
- HTML; supported with client and/or server side scripting.

You may also wish to choose a combination of the above—for instance a web-based interface for students and a stand alone program for the other users.

Your overall architecture and implementation platform choices are the most important decisions that you will make. The choices you make can have enormous impact on the ease or otherwise of completing the project. Don’t rush into this part of the project. Actually write down on paper the alternatives and list advantages and disadvantages of each from your perspective. Then you can evaluate objectively the alternatives and make some good decisions.

3.4 The Domain

We assume that you are familiar in general terms with the assessment process at a university.

The system must deal with various entities, such as:

- The student: must have a name, student number, and an email address. Students enrol in courses.
- An assessment item. It must have a name, a due date, maximum marks available, and a weight (what percentage of the aggregate mark is contributed by this item). An assessment item is associated with a particular offering (semester and year) of a course.
- A course has a name and course code.
- A course offering is a particular instance of a course, in which a student can enrol. So an offering is defined by course code + year + semester.

3.5 System Functions

PostMark must allow the following to be done.

1. Create/edit/delete a course. You cannot delete a course for which details of a particular offering have been described.
2. Create/edit/delete a course offering. You cannot delete a course offering if there are students enroled in that offering.

   Creating a course offering requires the specification of many parameters. For instance
   - The number and dues date of assignments
   - Maximum marks and weighting of assessment items
   - The names of the markers involved
   - Grading scheme (§3.6)
   - Late penalty rules (§3.6.1)
3. Enrol/withdraw a student from a course offering.
4. Enter a mark for a student for an assessment item. If it is a late submission then either the number of days late, or the date of submission, must be recorded.

   If a mark already exists for that student it should not be deleted. A history or log of (date + mark entered + user who entered the mark) should be maintained. Normally only the most recent would be displayed.
5. Enter a textual comment on a per-student or per (student + assessment item) basis. This records some information for later use by the examiner.

6. Record (with a date) that a particular assessment item for a student has been passed to a marker for marking.

7. Record that an extension has been granted, and how many days. This is on a per-student/assignment basis.

8. Automatically assign grades for the entire class based on the current rules (see section 3.6). This could occur for the entire class on request from the examiner, or be automatically calculated for individual students whenever a new mark was posted to the system.

9. Manually over-ride a system-generated grade. The system would not then be able to update this grade with a calculated value.

10. Remove the manually allocated grade, thus allowing the system to allocate a grade.

11. Release grades to students. Students should not be permitted to see their final grade until all grades for students have been finalised. See item 14 below.

12. Show *all* details for a student, for a particular offering. This would include all student details like name and student id, together with marks posted, status of not yet marked assessment items, and comments that have been entered into PostMark.

13. Produce reports (these are created as a text file for viewing on screen or submission to a printer). A report will be for a single offer (course+year+semester). Here is a minimum selection:

    (a) All students, showing student id, name, raw marks for all assessment items, overall percent for non-examination items, overall percent for examination(s), aggregate percentage, and suggested grade.

    This should be available as either:

    - sorted by aggregate mark (descending order)
    - sorted alphabetically by family name

    (b) All students, showing student id and grade, sorted by student id.

    (c) By marker: showing outstanding assignments that have been distributed to the marker but for which no mark has been recorded. For each marker the report should show: assignment number, student id, student name, days since it was given to the marker. Sort by assignment number within marker.

    You are encouraged to produce other reports that you believe would be useful.

14. Allow students to access marks and grades. Marks for assignments and examinations should be available immediately that they have been posted to the system. Final course grades are only available after being released by the examiner.

    A student should not be able to any other student's results. Hence access must be password protected.

3.6 Calculating aggregate marks and grades

The following describes the calculations required for a particular course offering. Remember that the number and characteristics of assessment items can vary between offerings.

Let there be n assessment items. For any assessment item i, where $1 \leq i \leq n$, the maximum marks available is M_i and the weight or contributing proportion of the item is w_i, which is expressed as a percentage: $0 \leq w_i \leq 100$. The marks awarded for a student's assessment item is m_i, where $0 \leq m_i \leq M_i$. An assignment mark as recorded may be reduced if the assignment is late. Let the new mark be m'_i; it is calculated as described below (section 3.6.1).

The aggregate mark a is calculated by:

$$a = \sum_{i=1}^{n} \frac{m'_i}{M_i} \times w_i$$

Grades are based on aggregate marks. The grades available are HD, A, B, C (pass grades) and F (Fail). (Note that this is a simplified system compared with a "real" university grading system which would allow incomplete grades for those who have not completed all assessment items.)

Four values define the dividing points between these 5 grades. The values can be varied between courses, but a common set are 50%, 65%, 75%, and 85%.

A student who has earned a pass grade based on the aggregate mark will be awarded a fail grade if she/he has not satisfied other criteria that apply for the offering of the course. Some of the rules that are commonly applied, and which PostMark must accommodate, are listed following. Note that one or more of these rules may apply to a given offering.

- The student must submit all assessment items.
- The student must achieve at least X% in the examination(s). (X is usually set at 50%, but may vary.)
- The student must gain at least X% of the weighted marks available from assignments (or non-examination items). (X is usually set at 50%, but may vary.)
- The student must gain at least X% in each assessment item. (X is usually set at 50%, but may vary.)
- The student must gain at least X% of the weighted marks available from assignments (or non-examination items) **and** must submit all assignments. (X is usually set at 50%, but may vary.)

Other rules may be required in the future. Your system should be designed to integrate new rules with minimum effort.

3.6.1 Late penalties

Late assignments may be subject to penalties. These are applied on a per-day late basis. There are a number of variations that PostMark must handle. Offerings differ in the scheme applied. There are three parameters involved here:

1. The percent per day deducted.
2. Whether deductions are made per calender day (including weekends) or per working day (Monday–Friday only).
3. Whether the deduction is a percent of the *mark awarded* to the student or is a percent of the *maximum mark* possible for the assignment.

Let the days late be k, and the per-day deduction be $x\%$. Let maximum and actual marks be M and m. If deduction are based on the awarded mark then

$$m' = m - \frac{m \times k \times x}{100}$$

otherwise if deductions are based on the maximum mark then

$$m' = m - \frac{M \times k \times x}{100}$$

The modified mark m' cannot be negative. If the above calculations yield a negative value then use a value of zero instead.

3.7 General

Try to make this a “user friendly” and flexible piece of software. Observe basic principles of building intuitive, usable interfaces as discussed in texts on user interface design and our own course CSC3402 GUI Programming.

Remember this: the person marking your work will not have read in detail, and does not wish to have to read in detail, a user manual in order to undertake most of these functions described. So you must make the interface clear, simple and intuitive. Assume little prior knowledge.

By flexible we mean that the system should not have “hard-wired” limits like number of students per class, or fixed percentage cut-offs. This will likely mean that information is held in offering-specific configuration files, rather than being embedded in the program.

References

[1] Alan Dix, Janet Finlay, Gregory Aboud, and Russell Beale. *Human-Computer Interaction*. Prentice Hall, London, 2nd edition, 1998.

[2] M. Fowler and K Scott. *UML Distilled: A Brief Guide to the Standard Object Modeling Language*. Addison-Wesley, 2nd edition, 2000.

[3] I. Jacobson. *Object-Oriented Software Engineering: A Use Case Driven Approach*. Addison-Wesley, 1992.

[4] K.E. Kendall and J.E. Kendall. *Systems analysis and design*. Prentice Hall, 4th edition, 1998.

[5] Leszek Maciaszek. *Requirements analysis and system design*. Addison-Wesley, 2001.

[6] R.S. Pressman. *Software Engineering: A Practitioner's Approach*. McGraw-Hill, 3rd edition, 1992.

[7] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, and W. Lorensen. *Object-Oriented Modelling and Design*. Prentice Hall, 1991.

[8] Ian Sommerville. *Software Engineering*. Addison-Wesley, 5th edition, 1995.

[9] Ian Sommerville. *Software Engineering*. Addison-Wesley, 6th edition, 2000.