CSCI222
Spring Session, 2015

Assignment 1: "Elaboration and Construction"
20 mark group assignment

Aims

The aims of this assignment are to:

- Provide experience in structured group work;
- Provide experience with the use of UML models that guide program implementation;
- Provide some limited experience with a simplified version of the Rational Unified Process (RUP) approach to software development;
- Provide experience with the disciplined use of a group-oriented software versioning system;
- Exploit "unit testing" frameworks in program development.

Objectives

On completion of this assignment, you should be able to:

1. Explain the phases of RUP and its iterative approach to software development;
2. Identify the standard roles that RUP identifies for participants in a software project, characterize their workflow patterns, and specify the artifacts for which they are responsible;
3. Participate effectively in a RUP project working in one or more of the identified roles;
4. Explain the nature and use of UML's "use case", "class", and "sequence" diagrams for modeling aspects of a software system; interpret UML “activity” diagrams;
5. Participate effectively in an iterative software development process, contributing implementation elements that are integrated to create the overall application;
6. Utilize a version management system that supports group work;
7. Perform unit testing, integration testing, and system testing.

All coding will be in C++. The cppUnit framework will be employed for unit testing. The default code versioning system is *subversion* deployed in combination with an Apache web-server; other version management systems may be used. (Note that some of the “objectives”, e.g. “explain the phases of RUP …”, relate to the overall objectives of the subject, and to the examination assessment. This assignment is your chance to “play RUP” and so gain some practical understanding of this software development process.)

Tasks

Your tasks are to:

1. Form and structure your group, allocating roles and responsibilities to your members;
2. Complete the design and implement the project described in the attached document;
3. Produce a report detailing the group's work.

The "software process" elements for this assignment may seem rather heavy given the limited scope of the actual application that is to be developed. This assignment is about practicing processes that will be useful in larger scale exercises; the product developed is of no great consequence.

Of course, you do have to create a working program in order to prove the effectiveness of the process that you followed! The project description identifies a number of possible iterations in construction; these iterations provide increasing functionality in the application. If your group works slowly and fails to implement all iterations, your group mark will be reduced.

Relatively few marks are allocated to the product (you have to be able to demonstrate a working application for a couple of marks, and the quality of your C++ code is assessed and this assessment forms part of the mark for the report). Instead, the marking scheme allocates marks to evidence for a serious attempt at following a RUP style development process along with attempts to make use of support tools like subversion, and unit testing.

Individual marks are based on the group mark. Individual marks are adjusted in accord with the “member’s contribution assessment” document submitted by the group.

Group and roles

The roles required for this group assignment are:

- Manager (1)
- Lead designer, lead implementer (1) : probably the same person, but the "lead" role could be split;
- Designer (many)
- Data persistence specialist (1)
- Implementer (many)
- Systems integration and systems test (1)
- Documentation (many).

The designated group leader should take the role of manager or of lead designer/implementer. Each implementer is responsible for unit testing of his/her code.

The book *The rational unified process: an introduction* by P. Kruchten contains explanations of the RUP roles. Details include lists of artifacts that are the responsibility of a person holding a particular role, and workflow diagrams. Students should check these outlines to gain guidance as to what they are to do in the project. The Kruchten book is available as an electronic resource through the University library's subscription to Safari.

Each group will hold formal minuted meetings. The group member holding the "manager" role is responsible for these meetings, in particular preparing agendas and keeping meetings on track, and is also responsible for the reports on meetings. The meeting structure should be along the following lines:

- Reports from individuals who were allocated "urgent action items" in the previous meeting; discussion of issues related to these "urgent action items";
- Manager reviews his/her perception of the current status of the project; this review is to use data from bug reports and feature tests as created by the systems integrator;
- Scheduled agenda items are discussed - in the early meetings this will involve resolution of open design issues, later it will involve review of progress on implementation elements and integration;
- Reports by individual members on progress in their areas of responsibility ("code walkthroughs" may be used)

and identification of any problems;
- Identification of any "urgent action items" and allocation of responsibility for these;
- Proposals for agenda items for the subsequent meeting.

Each group member will maintain a work diary. This should cover: planned work schedule, actual work times, summary of work completed, report on "defects" in own work. The purpose of a work diary is really to provide data that identify ineffective work practices so that remedial action can be taken. Often you will find planned and actual work times differ significantly; such differences may point to poor work management practices. Every developer has his/her own blind spots that lead to defects in code. Analysis of defect patterns can help identify weaknesses and thus lead to a pre-emptive approach where the developer checks for their typical errors prior to running code.

Design and implementation issues

The system design, as given in the accompanying document, is partially done. It identifies main activities, use cases, and some potential classes. (*The document may seem long. In fact, it is abbreviated. I have included sufficient for you to see samples of diagrams and for you to get some feel for how these "models" will help coding. The structure of such a design document varies with the "ceremony" level of the design process, and also will vary from workplace to workplace. A "vision" is generally considered useful – it's meant both to give a very high level view of the aim of the project and in some way act as an inspiration to developers. The "business case" for the project explains how the company developing the project (or their customers) will benefit from the implemented computer system. You will then get some listing of functional and non-functional requirements together with priorities. These requirements will frequently map one-to-one onto use cases – the ways that your users will work with your system. A design document will, hopefully, identify all the major use cases – but there are always extra ones added later. A use case shouldn't be simply a "stick-man diagram"; it should contain some real information to guide the implementers. The system architecture should be presented, along with at least some discussion of data persistence. The document provided to you is meant to correspond to something created part way into "elaboration", so it also includes some initial ideas as to how particular use cases will trigger a series of interactions amongst class instances. A design document should include some planning data showing how the work will be completed in subsequent elaboration and construction iterations.*)

One of the first tasks for your group will be completion and refinement of the class definitions that are supplied. Your group might chose to rework the class identification steps and derive a different set of implementation classes. You might want to refine the informal descriptions of how processing is done into more formal "sequence diagrams".

The specification includes an "iteration plan". Your group may wish to revise this plan; you are required to use an iterative approach with more than one iteration in the "construction" phase.

Report

The work of the group is to be presented in a report. Overall responsibility for the report lies with the person in the "manager" role, but each member will contribute.

The report is the primary basis for your assessment. The report should be concise; reports that are excessively long will be received unfavorably by the markers. The report should briefly present the product that you created and provide evidence for a competently handled development process. It should use grammatically correct English and should not contain innumerable spelling errors. You should use the spelling and grammar checkers in your word processor application.

The report could be structure along the lines indicated below. Your group may choose to adopt a different structure if you wish; your modified report structure will still need to cover the aspects listed below.

- A brief overview of the project based on information in this assignment specification and details of the actual application. This overview should not be created by simply cutting and pasting text supplied. It should be written by the members of the group and provide a retrospective view of the project on its completion. It should be "complete" in the sense that it explains the project to an external assessor who has not seen any of the assignment details.
- A brief presentation that illustrates the actual implemented product. This presentation should combine text commentary with information captured from actual execution of programs (as screen shots or as captured text inputs and outputs). *This presentation should clarify which of the functional requirements have been successfully implemented.*
- A tabular summary of the group structure identifying group members, the roles that they filled, the artifacts that they successfully delivered.
- A summary of your group's work on design and the implementation plan. This should cover: any reworking of the proposed implementation

classes and give details of decisions relating to data persistence and user interface issues. UML modeling diagrams should be used to illustrate design decisions. If your group decides on a different implementation plan, with different iterations, you should give details and justification.
- Details of the construction phase. This part of the report should clarify the work done in each iteration.
  - Summarize the new elements added and the extensions to existing implementation elements. Include brief details of unit testing procedures used to verify new elements prior to their commitment to the project (this does not mean list unit test code and test outputs, simply identify the additional tests created by individual implementers).
  - Summarize data from defect and integration reports created by the systems integrator.
  - Provide evidence for the appropriate use of version control software; this would typically take the form of excerpts from subversion's logs of commit operations.
    Subversion statistical reports, showing overall contributions by different members, could be included in the report on the final iteration.
    N.B. some practice use of the subversion code management system is a requirement of this exercise. Of course it's overkill here. The intention is that you practice the use of such technology on something fairly trivial before you need to use it for real as in CSCI321.
    It is permissible to use a different subversion server, or use an alternative version management system such as GIT. The important thing is that you gain some practice in the use of a version management system.
- Group meeting records and individual diaries:
  - There should be a summary detailing the work done at each formal group meeting.
  - There should be an example agenda, and report from at least one of these meetings.
  - There should be samples taken from the work diaries of at least two members of the group.
  - There should be samples from bug logs and testing logs
- Code
  - Code listings for all elements in the final product should be included in an appendix to the main report.
  - Samples should be provided of support code such as the unit test classes created to verify particular application classes.

Submission

Your group is to submit a PDF formatted report file. This report file should be prepared in a word processor such as Word and "printed" to PDF file. (Note, a well prepared report on this task might be a useful addition to the "work portfolio" that you are preparing to show potential employers.) The designated group leader is to submit this report electronically using the turnin system. (Don't have multiple members submitting different versions of the report!)

The due date for submission will be announced in lectures; the date will probably be around the end of week 9 of session (currently set for Friday September 25th).

```
turnin -c csci222 -a 1  A1.pdf
```

In addition to the report, your group is to submit, in hard copy, its "member contribution assessment" document. This should identify the member contributions on the ordinal scale "contributed", "some contribution", "almost no contribution", "no contribution". This document should be signed by all group members. This document should be submitted in the Tuesday lecture of week 10. (This "member contribution assessment" isn't required if the group is happy to have all members receive identical marks.)

Reminders:

1. turnin isn't chatty you don't get email back
2. You can check that your submitted file is safely submitted via the turnout program.
3. You should check that your uploaded PDF file is readable; marks are lot for corrupted PDF files.
4. turnin only works if you are logged in to banshee - use ssh if you are on a different machine or workstation. (If turnin complains that it doesn't know about the assignment, you aren't logged in on banshee!) You can use the Unix command `uname –n` to determine which computer you are logged into.

Marks

Your group's maximum mark:

- If your group is unable to implement any of the specified functionality, your mark is zero.
- If your group implements only the functionality identified with the first iteration, your group's mark is the smaller of 7 (seven) or the mark actually awarded for your report.
- If your group implements the functionality associated with the first two iterations, your group's mark is the smaller of 13 (thirteen) or the mark actually awarded for your report.
- If you implement the complete functionality, the report mark is the group's mark.

Mark breakdown

- Presentation of report
  4 marks
  This is a fairly large fraction of the overall mark. Much of "Software Engineering" is really just a matter of communication with your stakeholders. Your markers are stakeholders. They want a report that is complete, clear, *and concise*.
  Factors contributing to this portion of the mark will include: the clarity of the "retrospective project overview", the overall structure of the report document, appropriate use of embedded images (modeling diagrams, screen shots, subversion reports etc), layout of sections (particularly code layout), indexing of the report, clear sectioning, English grammar and spelling, and general appearance.
- Design and planning
  3 marks
  These marks are for your group's work on completion of the design supplied, particularly your work on the data persistence and user-interface aspects.
- Lab demo
  2 marks
  Convince a tutor that it really does work and work well with a good demonstration.
- Project management
  5 marks
  These marks are based on evidence for the use of an effective software development process.
  Factors contributing to this part of the assessment will include: evidence for an effective group structure and adoption of roles; a well defined iterative construction phase; effective group collaboration as evidenced by meaningful group meetings; disciplined development processes of individuals.
- Code
  6 marks
  This part of the assessment will be based on the quality of the C++ implementation code, and evidence for testing such as the unit test code and reports.
  The code should, of course, use properly organized header and implementation files for classes; good C++ coding style is required; elements written by different implementers should be consistent in style (the group should adopt conventions for naming and structuring of code that are used by all implementers). The code should not contain errors. Control flow should be well structured.
  (All those trace statements – "cout << "Start function insertnew" << endl; – they may have been essential during development, but remove them from the final version!)