University of Toronto

Faculty of Arts and Science
Dept of Computer Science

CSC302S – Engineering Large Software Systems

April 2008
Instructor: Steve Easterbrook

No Aids Allowed
Duration: 2 hours
Answer all questions.

Make sure your examination booklet has 10 pages (including this one). Write your answers in the space provided.

This examination counts for 35% of your final grade.

Name: ____
(Please underline last name)

Student Number:____

Question Marks

1______________	/20
2______________	/20
3______________	/20
4______________	/20
5______________	/20
Total __________	/100

1. [Short Questions; 20 marks total]

(a) [Software Quality – 5 marks] What does "quality" mean with respect to software, and how would you measure it? Why is it difficult to predict the quality of software before you have finished building it?

Quality can be defined in several ways: E.g. fitness for purpose, value to some person, exceeding the customer's expectations. To measure it, you really need to assess the software once it is in use, and ask users (or observe them) to see how well the software is helping them to do whatever activities it is intended to support. To do this systematically, you would normally determine a set of quality goals from the requirements (e.g. performance, usability, reliability, etc), and then identify specific measureable attributes that correspond to them (e.g. time taken to perform some representative task).

Quality is hard to measure before the software is finished because quality is an emergent feature of the software as a whole, and depends on how the users experience the software in a particular context. Measurable quality attributes help to predict the quality of the finished product, but cannot accurately predict the user's reactions to it.

(b) [Verification Strategies – 5 marks] What is the difference between testing and static analysis? What are the advantages and disadvantages of each of these approaches to verification?

Testing finds defects by running the program and checking that it does what it is supposed to do. Static analysis finds defects by analyzing the code (source code and/or object code) and looking for particular kinds of programmer error. Static analysis does involve running the code.

Testing has the advantage of explicitly comparing the program's behaviour to a specification of what is expected, and it simulates what the users will experience. However, the major disadvantage is that it is expensive – each test takes time to run, and a huge number of tests are needed for to cover the different behaviours of a program. Exhaustive testing of all possible paths through the code is usually impossible.

Static analysis has the advantage that it is relatively cheap, because you don't have to execute the code, and it concentrates on finding common types of error. It can also find errors that are hard to find in testing, such as memory leaks. The major disadvantage is that it only finds the kinds of errors that it has been programmed to find, and can't tell you if the program does what it is supposed to do.

(c) [Requirements Analysis – 5 marks] Why is it useful to write down the requirements for a software project before attempting to design a solution? Why is it hard to do this well?

Writing down the requirements in advance is useful because it helps to pin down what problem needs solving and so avoid wasting time building software that doesn't do what the customers want it to. It also gives clear statement of purpose that will help assess whether the implementation is correct, so can be used to generate test cases, etc.

It's hard to do this well because the requirements are continually changing as the users' needs evolve, and as the environment in which the software will be used evolves (e.g. changes in technology, changes in the business environment, etc). Also, it is often hard find out what the problem is, until you attempt to solve it.

(d) [Agile Development Practices – 5 marks] Agile practices are often considered to be appropriate only for small teams (e.g. less than 10 people). Which aspects of agile development can still be used effectively with larger teams, and which aspects fail to scale up?

[Many possible answers. Suggestions include:]

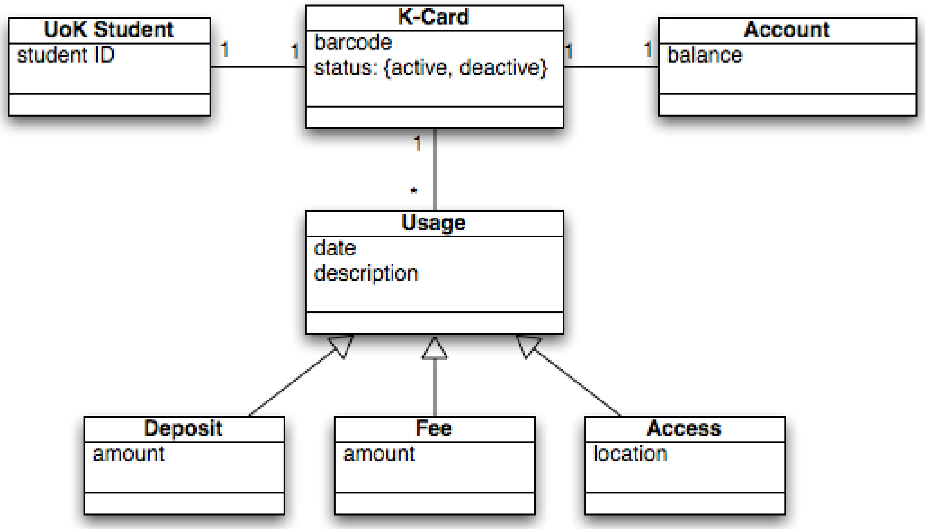
Key features of agile development include the emphasis on human communication over documentation, and giving developers much more freedom over how to organize their work. These aspects are hard to scale up because in teams much larger than 10 people it is hard for everyone to communicate well with everyone else (huge number of communication paths), and so teams need more explicit coordination mechanisms and a more explicit record of what they are doing.

On the other hand, ideas like small iterations, test-case first, and pair programming (to get immediate peer review) can be used in any sized team.

Some practices have to be carefully adapted. For example, agile practices tend to avoid "upfront" design, relying instead on extensive refactoring if the initial design choices turn out to be unsuitable. This is harder to do on larger projects because of the cost of rework, but some occasional refactoring is still useful as the design choices can still turn out to be unsuitable now matter how much upfront analysis was put into them.

2. **[Black Box Testing – 20 marks]** You have been asked to test a program that takes as input three integers that represent the lengths of the sides of a triangle. The program outputs a string giving the type of triangle: “Equilateral” (if all three sides are equal), “Isosceles” (if 2 sides are equal), “Scalene” (if no sides are equal), or “Error” (if the inputs cannot be a triangle). What test cases would you use to test this program as thoroughly as possible? What else would you need to know in order to improve the thoroughness of your test suite?

3. **[Class Diagrams – 20 marks]** The University of Kinakuta (UoK) has decided to implement an electronic card system for its students, so that students can use their K-cards to access secure areas (.e.g labs and residences after hours), and also as a debit card, linked to an account into which students can deposit money, to be used to pay University fees at locations on campus. For the initial release of the system, this will be limited to a few university fees: parking fees at campus parking lots, library fees at campus libraries, and equipment rental at the sports centre. The system will keep a usage record for each K-card. An initial domain model for the system has been produced:



(a) [5 marks] In the model provided above, there is a 1-to-1 association between a student and a K-Card. Is this an appropriate way to model students and K-Cards or is there a better way? Defend your answer.

(b) [5 marks] In the model above, there are no explicit associations from the Deposit and Fee classes to the Account class into which these transactions are to be made. Should such associations be added? Defend your answer.

(c) [5 marks] In the original model, the relationship between the Usage class and K-Card class is an association. Alternatively, this relationship could be modeled using an aggregation or composition. Which alternative is better for this problem domain and why?

(d) [5 marks] After further discussions, UoK decides to add more features to the system. Food services wants to keep track of each student's meal-plan allowance when they use their K-cards to buy food and snacks at all campus eateries. Computing services wants to keep track of each student's printing allowance, when they use their K-cards at self-service printers across campus. The meal-plan allowance and printing allowance are separate from the debit-card balance. Student will also be able to use their card to make purchases (as well as paying fees) and to make deposits at many locations across campus, and the system will need to keep track of the location of each transaction. Revise the original domain model to allow for these additional features. Cross out elements that are no longer needed, and add any new elements. Use multiplicities, associations, generalizations, etc as appropriate. You may use the space below to re-draw the entire model if necessary.

4. **[Topic – 20 marks]** Describe ***two*** examples of software engineering techniques (or tools) that you used on your course project this term: *one* that was particularly effective, and *one* that was not. In each case, explain how you used the technique (or tool), and the factors that caused it to be useful (or not). Your answer should take into account a consideration of whether the technique/tool was suitable for the type of software you were building, whether you used it at the right point in your development process, and whether you had the right expertise to use it properly.

Many possible answers. For full credit, must expain what the tool/technique was, and address all of: suitability, timing and expertise. Plus must say clearly why each was effective or not. Answer must show good insights and reflection on the software development processes to get full marks.

5. **[Software Communication – 20 marks]**. Imagine that a newly hired programmer is to join your team to assist with the work you are doing of adding new features to the UMLet application. Draw some pictures to explain to your new teammate the structure and behaviour of UMLet, and the feature(s) you have added in your course project. Your pictures should concentrate on key aspects of the design that would be most useful for the new team member. Assume your new team member is familiar with all of UML, but knows nothing about the UMLet application.

Many possible answers. To get full credit, the answer must include both structural information (e.g. class diagrams, component diagrams, architecture etc), and behavioural information (e.g. sequence diagrams, etc), and must comment on what these diagrams show and why they were chosen. Credit is given for identifying key fetures of the design, and using UML (or other kinds of diagrams) appropriately. Also the diagrams must show something about the new feature(s) that were added, as well as the original version of UMLet.

[scratch paper]

[scratch paper]