Information and Computer Science 52
Introduction to Software Engineering
Winter Quarter 2004

Name	___Solution Key and Grading Guide_____
Student ID	______________________________
Signature	______________________________
SEAT #	______________________________

Instructions

1. This exam is closed book, closed lecture notes, and closed personal notes. No PDAs, cell phones, SMS devices, … You may use an English/non-English language dictionary.

2. This exam consists of 10 questions on 6 pages; answer all questions on all pages.

3. The questions are **not** equally weighted. The relative point value of each question is given at the beginning of the question. The total numbers of points that can be earned on this exam is 65.

4. Write your name, student ID, seat number, and sign the exam where indicated above. In addition, write your name on **each** page of the exam. Failure to do so results in a loss of points.

5. Before you cheat, consider the ramifications: you will score an F for **all** of ICS 52, not just the midterm.

6. Before you cheat, consider the chances: multiple eyes will be watching you from multiple locations in the room.

7. All answers are to be legible. Write clearly.

8. All answers are to be given on these pages. Use the backsides of the pages **only if necessary**, with a reference as to where any extra portion of an answer is located.

9. You have one hour and fifteen minutes for this exam – that is, until 3:15p.m. Plan your time accordingly.

1. (10 points) AOL does not use a peer-to-peer architectural style for their chat application (AIM). For the purposes of this question assume that the architecture is client-server (it's actually a bit more complicated than that.)

   (a) What advantages are there, from AOL's perspective, to the client-server approach? You should briefly (i.e. one sentence) describe 3 or 4 such advantages.

   1. The clients are kept simpler, and therefore have the potential for being installed on many different platforms more easily.
   2. If changes are made to concepts like buddy-lists, they only have to be implemented/changed one place, namely on the server.
   3. AOL could decide to start charging for the service and they could force that to happen by limiting access to the server.
   4. AOL could decide to offer "premium services" similar to #3 above.
   5. AOL can keep performance adequate by deploying more hardware on the server side, as opposed to telling the clients that they need to upgrade
   6. Privacy may be enhanced for the users, e.g. for chat applications. Only the server knows the addresses of all the chat participants. In the p2p situation each member of the chat room could know the IP addresses of all the other participants.

   (b) What advantages would there be to switching over to a peer-to-peer (p2p) style? Would those advantages be primarily for the users? For AOL?

   1. (user) The users would be invulnerable to changes in AOL's pricing, policies, services
   2. (both) The system would be more robust against failure
   3. (AOL) The cost of supporting more users would be borne by the users, not AOL.
   4. (user?) With all the extra complexity residing in the peers it wouldn't be long before someone figured out how to make more interesting use of those services than just supporting chat!

2. (15 points) They just don't give up! Your company, Crazy Software Ventures (CSV for short), is at it again. The company's management has been watching the news about the Mars Exploratory Rovers (MER), Spirit and Opportunity, ever since they landed on Mars in December and January. They see a business opportunity. Recall that Spirit went virtually dead for over a week, due to a software problem. They've decided to try to convince the JPL managers to abandon in-house software development and contract out all the software development for the next Mars Rover (to be called "HIgh SpiritS" or HISS, for short). Naturally CSV hopes to win this contract. If they do, you'll be in charge of the development project for HISS!

   (a) What software development process ("lifecycle model") will you adopt, and why?

   Perhaps more than anything else, the quality of the software has to be the finest there is. The whole objective is to avoid embarrassments such as flying into a planet, forgetting to collect your garbage (files), and so on. So the process has to be one that values quality above all else.

   The process also has to deal with a sharp separation between the customer (JPL) and the engineering company (CSV). So there need to be clear points at which both sides sign off on key decisions.

My choice would be the spiral model. It addresses risk, risk, risk, and there's a lot of that on the way to Mars. Yes it is expensive, but it is cheaper than wasting a Delta II booster on the launch pad!

(b) What will you emphasize in the requirements phase(s), and why?

1. Making sure the interfaces to the hardware are very well understood, and accurate. An error here could mean that the software is "correct" but that HISS still tips over.
2. Making sure that the robotic engineers' models for how HISS will be controlled are effectively and accurately described. Why? I don't want the engineers yelling at me to say that they did their part right and I just didn't understand it.
3. Consistency! The last point I want to discover that I have contradictory requirements is on Mars. Or even during the development phase.
4. Ability to accommodate change. I've heard stories about software being uploaded mid-flight, so I'm going to need to be able to assess the impact of requirements changes that come up later on.
5. Clarity. A **lot** of people are going to be looking at this specification, and I want them to all understand it well and accurately.
6. A great glossary. I'm no rocket scientist, so I want the rocket scientists to tell me exactly what they mean by things like "altitude" and "nominal".

(c) What will be the priorities for your architecture (i.e. the key characteristics or properties), and how do you expect to achieve them (i.e. what style(s) will you use; how will you create the architecture)?

1. Dynamic update. I just know that I'm going to have to update parts of the system, probably while HISS is on its way to Mars. So I better have an architectural style that helps me do that.
2. Ease of analysis. I really want to have confidence that this program will be **right**. So if the architecture is too complex or too clever, I'll be in trouble.
3. Ease of understanding. A lot of people will be looking over my shoulder…
4. Ease of testing. For the quality reasons described above.

3. (5 points) What are the five (or six, depending on how you count them) "Recurring, Fundamental Principles" of software engineering? (Here's a hint: the first one is "rigor and formality"). For each principle name it and give a one-sentence description of the principle.

a) Rigor and formality: absence of imprecision and informality.

b) Separation of Concerns (Divide and conquor)
Modularity: separation into distinct physical part
Abstraction: showing only those aspects that are necessary for the user of a concept, and hiding the details only of concern to the "implementor"

c) Anticipation of change: thinking ahead regarding how the concept at hand might change.

d) Generality: avoiding overly specific approaches. Seeking to identify abstractions useful in more than one circumstance.

e) Incrementality. Doing, and delivering, in a step-wise fashion to foster feedback and analysis.

4. (5 points) Complete the following definition of the uses relationship: "Module A uses module B ….

   … If and only if the correct execution of B is necessary for A to meet its specifications.

5. (5 points) Which desirable property of requirements documents, completeness or consistency, is more important, and why? If your answer depends upon the perspective (namely, "customer" or "developer") or upon the application domain (e.g. "financial accounting software", "video games", …) then explain how the perspective and/or domain determines your answer.

   LOTS of possible answers here. Customer: "I'm interested in completeness. I want to be sure that the developer is going to give me everything I really need. Completeness? That's something those smart engineers can figure out." Developer: "Consistency. The last thing I want to be contractually obligated to do is build something that has internal contradictions!" Financial accounting software: both are equally important. If I cheat on either one somebody is going to either sue me or steal from me.

6. (4 points) Software engineering techniques and processes are not appropriate for all development projects. List at least four characteristics of a project that would argue AGAINST using the software engineering approach.
   - Very Small project
   - Single developer
   - You can build what you want… you don't have any customers
   - One product; not a family
   - Short-lived … I'll throw it away soon
   - Cheap. I can't afford to do a good job ☺
   - Small consequences when something goes wrong.

7. (10 points) It just won't go away, this question is still here! List the 10 architectural idioms (styles) covered in class and provide a brief (one sentence) description of the key characteristic(s) of each style.

   1. Batch sequential .
   2. pipe-and-filter
   3. Data and/or service-centric systems: the Client-Server style, a.k.a. database centric design
   4. Hierarchical systems/Main program and subroutines;

5. Data abstraction/OO systems
6. Peer-to-Peer
7. Layered systems
8. Interpreters
9. Implicit invocation (event-based)
10. Three-level architectures

I'll refer you to the lecture notes for the descriptions.

8. (3 points) Describe three ways in which the Waterfall process model is superior to the Build-and-Fix model.

    1. clear phasing of many activities (requirements, design, coding, testing)
    2. a sense of direction: knowing from the outset what the goals are
    3. appropriate for large-scale projects
    4. allows many specialists to contribute to their part
    5. explicit design, separating the concerns between low-level programming stuff and the real work of determining the system's structure
    6. uses intermediate documents for communicating between the phases.
    7. other answers possible…

9. (4 points) What is a module's *provided interface*?

    It is the description, in precise terms, of everything another module needs to know in order to make use of this module's services. Or: it is the precise description of the services offered by a module.

    What is a module's *required interface?*

    It is the description of (or references to) all the services that other module's must provide to me in order for me to get my job done.

10. (4 points) Why should the developer be interested in creating an excellent acceptance test plan?

    Sample answers:
    a). It provides a clear way for me, the developer, to demonstrate to the customer that I have done my job well and properly, and hence deserve to get paid.
    b). It can help me keep focus on what I'm supposed to be building.
    c). I can use it measure my progress towards completion, when I'm in the final stages of checking out my system.

    Why should the customer be interested in having an excellent acceptance test plan as part of the requirements document?

    It is the way I can tell whether or not the developer built what I am looking for. Since I don't understand code, or designs, or architecture, it is the one thing I can do to assure, in terms that I understand, that my needs have been met.