Information and Computer Science 52
Introduction to Software Engineering
Fall Quarter 2001

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, …
2. This exam consists of 12 questions on 7 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 63.
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 4:45p.m. Plan your time accordingly.

1. (8 points) Software engineering techniques and processes are not appropriate for all development projects.
   (a) List at least four characteristics of a project that would argue FOR using the software engineering approach.

   Grading: 1 point per characteristic

   Example characteristics: Size of project (big), number of people involved (>1), external requirements imposed, family of products required, many changes over time requested or likely to be requested, costly, large/major consequences as a result of the development.

   (b) For each of the characteristics you list in part (a), describe at least one way software engineering attempts to address the concerns raised by that characteristic.

   Grading: one point per characteristic

Example answers: Size [applying software process to the development; breaking the problem into parts using modularity; …], Number of people [process], external requirements [requirements document, acceptance test plan, reviews], family of products [software architecture], changes [design by information hiding, software architectures], costly [process model, reviews, reuse], consequences [quality assurance techniques]

2. (6 points) Despite the fact that a pipe-and-filter style architecture can be developed for the Congo.com back office Order Fulfillment Process (OFP), it is a very poor choice of architectural style for that problem. Briefly describe three reasons why it is a poor choice.

Grading: 2 points per reason. Reasons 1 and 2 below are key, the third reason could reflect a lot of different ideas.

Example reasons:

(1) Lack of information hiding: multiple components have to manipulate the order record, but that data structure is not encapsulated
(2) Lack of info hiding: same as above, but answer focused on code duplication … e.g. for continually reparsing the input stream with the order records, or a second component that sends out email to the customer, with only slight differences from the first one.
(3) Doesn't seem a natural fit to the problem (which looks and feels like a client-server or database centric architecture)
(4) No obvious way this supports a program family
(5) Doesn't anticipate change very well

3. (3 points) Grading guide: 1 point per part of the question.
   (a) Why should all the details of an application's user interface be included in a requirements specification?

Because such details are externally visible, the user most likely cares about them, and such details DO describe what is required of the software.

(b) What makes it seem inappropriate for including those details?

Such details emerge from a design process, namely the user interface design process. Thus design would have to go on during requirements.

(c) Which software process model best supports development of applications with successful user interfaces, and why?

Several do. The Spiral model and any incremental model do. They work because the process allows users to state their initial goals and requirements, all the developers to work ahead a bit (do an initial UI design, e.g.), and then give the users the opportunity to review and rethink what they want.

4. (10 points) List the 10 architectural idioms (styles) covered in class and provide a brief (1-3 sentence) description of the key characteristics of each style.

Grading guide: 1 point per style. Half for the name, half for the definition. Rather than repeating the definitions here, I'll refer you to the lecture notes: first set of architecture slides (week 3), beginning with slide 13.

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

5. (3 points) Describe three ways in which the Spiral process model is superior to the Waterfall model.

Grading guide: 1 point per way.

Example answers: Explicit provision for risk analysis and mitigation; explicit provision for replanning of the process, in mid-stream; explicit consideration at all stages of alternative approaches; supports exploratory prototyping. Other answers possible.

6. (3 points) What are the three most important characteristics of a requirements specification? Explain why each of them is so important.

Grading guide: 1 point per way.

Completeness: if it is incomplete then additional requirements will be identified downstream, possibly upsetting the development entirely.

Consistency: if the document is inconsistent, then what should be built?

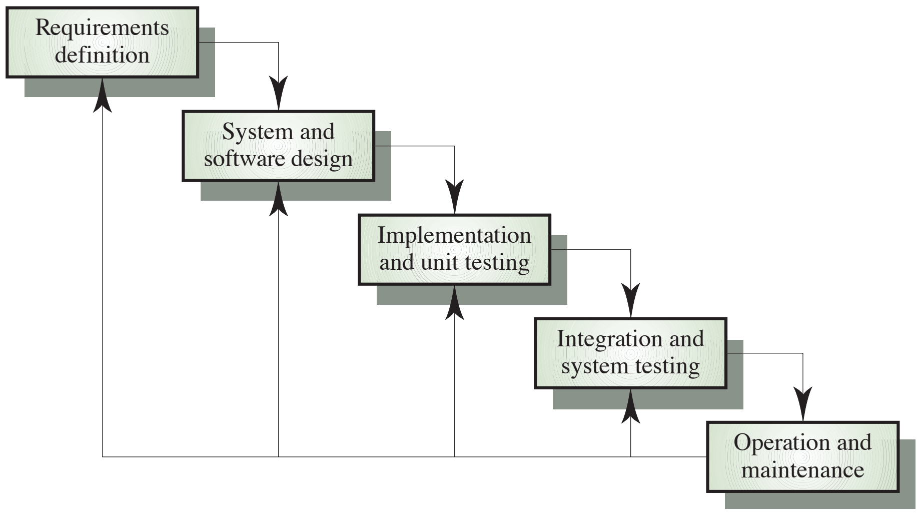
Unambiguous: if there is ambiguity a system might be built that does not satisfy the real requirements.

Other possible answers:	Verifiable
	No implementation bias.

7. (2 points) What's the point of identifying subset/superset systems in the requirements specification?

Provides the basis for developing a solution architecture that will be capable of supporting incremental delivery and incremental development: the subsets can be designed and developed (and deployed) first, without requiring that the whole system be built in one single activitiy.

8. (8 points) Grading guide: 2 points for part (a), 4 for (b), 2 for part (c)
(a) Describe the Waterfall model of software development. Use a diagram to illustrate your version of the waterfall model. (Variations on the following are possible)


Requirements definition
System and software design
Implementation and unit testing
Integration and system testing
Operation and maintenance

(b) Briefly describe at least four GOOD things about the Waterfall process model.

- Provides clear phasing of activities.
- Separates requirements from design/development
- Uses intermediate documents for communicating between the phases.
- Allows specialists to work in phases (as opposed to someone who has to work across all phases)
- It is simple
- There is provision for cycling back to previous phases
- (Other answers possible as well)

(c) Describe at least two ways in which the waterfall model is a POOR guide to software development.

- Requires all requirements to be complete before beginning design
- None of the benefits of the spiral model, as described in the answer to question 5

9. (2 points) What is the principal difference between a component's provided interface and its required interface?

The provided interface describes the services that component offers to the rest of the world. The required interface is a list of the services that module requires in order for it to be able to accomplish its job.

10. (2 points) What's the purpose of the acceptance test plan?

Provides a straightforward, unambiguous way for the customer to ascertain whether the delivered system meets the requirements. It can also be used by the developers to help assess whether they have met the requirements.

11. (4 points) What are the four essential elements of a software architecture?

Grading Guide: one point per element

Components: the locus of computation and state
Connectors: the locus of communication between components
Topology: how the components and connectors are hooked up…. their arrangement
Constraints: the rules that the other three have to play by in order to be legal.

(12 points) Your company, Crazy Software Ventures (CSV for short), has decided to develop a new word processor called BeyondWords (BW for short). In its desktop version BW will compete directly with Microsoft Word and similar products from other companies. BW is also intended for use on engineering workstations (running Solaris or Linux), and versions of BW will be required for PalmPilots and Compaq IPAQ's. Some of the people in CSV's marketing division think that a version of BW should even be available for cell phones. Whether or not this is an astute business decision, you have been named project manager. Your responsibility is to establish the detailed process by which BW will be created and lead the development team. Naturally your management wants BW on the market as soon as possible and they have a limited development budget. Grading guide: 4 points per section (a, b, c)

(a) Will you have an extensive requirements phase? Why or why not? What will you emphasize in the requirements phase?

You could successfully argue either side of this. Here's a couple of possible arguments.

(i) For extensive requirements. This is clearly going to be a tough market to enter: you're competing against MS in the desktop market, and trying to put functionality on some very limited platforms in other markets. It stands to reason, therefore, that you should very carefully consider the ways in which you might succeed. A careful requirements analysis might reveal specific features or capabilities that you could provide in these markets that would generate sales.. presumably features not found in MS Word. Or even if they are found in Word, if you can implement them in a small fraction of the space required by Word, you might win.

(ii) Against extensive requirements. Everyone knows what features a Word processor must have on desktop platforms, so why spend a lot of time documenting them? On the other hand, implementing such features on small platforms could prove a considerable challenge, especially with regard to their user interface/usability features. In this case, moving quickly into some exploratory architecture and generating some little prototypes might help reveal the key issues you have to focus on. So… you could call this requirements analysis in the sense of the Spiral model.

Note that in either case, your emphasis has to be on learning. Learning what features you must have, what features you can compete in the marketplace with, learning what are the killer issues in implementing BW on the various platforms.

(b) What will be the priorities for your architecture, and how do you expect to achieve them?

The problem statement highlights two things (at least) which govern the architecture choices. The first and most obvious one is the need to have an architecture that will support implementation on multiple platforms…. including platforms that have very different properties. This suggests a layered architecture, at the bottom at least, to insulate you from the vagaries of different user interface toolkits.

The multiple platforms, with vary different capabilities, also suggests that it needs to be a subsetable architecture. If you do go to the cell phone platform you won't need, and won't be able to tolerate, 99% of the features of your product that you'll have on the desktop version. So you need to think carefully about that. (While the cell phone is so extreme that you might have a wholly separate architecture for it, there is clearly a near-continuous spectrum of platform capabilities as you move from the Palm to the iPAQ to the small desktop machines to the engineering workstations.)

The second part of the problem statement that suggests some architectural priorities is the need to get an implementation out soon, and for low cost. Again this suggests a layered or subsetable architecture, in which you can concentrate on getting an initial, core product out the door early on. Good use of the uses relationship will help see you through this thicket.

(c) What qualities will you emphasize in the product and why?

My first choice would be usability. CSV is clearly targeting a wide range of users, but certainly "ordinary consumers" are part of the list. How will you win against MS except by having better (or at least different) functionality that is easier to use?

I would tend to downplay quality, in the sense of correctness, initially. This somewhat goes against the grain, but this is a high-risk venture (that's why the company is known as CSV), and if you're going to succeed it is going to be because of good buzz --- great features that are easy to use. Besides, customers are used to MS failing at unpredictable times in inappropriate ways, so you could get away with some of the same low-quality. (Alternatively you could play the "this is rock solid, anytime, anywhere" game. But you're likely to run into problems with your management --- who want the product out the door in a hurry, or technically, since this will be hard to achieve, especially on a platform like the iPAQ.)

Part of this is also likely to be speed. If you have a slow running editor then no-one is going to buy it.