Midterm Exam and Sample Solutions
CSE403 Summer 2006

July 24, 2006

NAME: _____

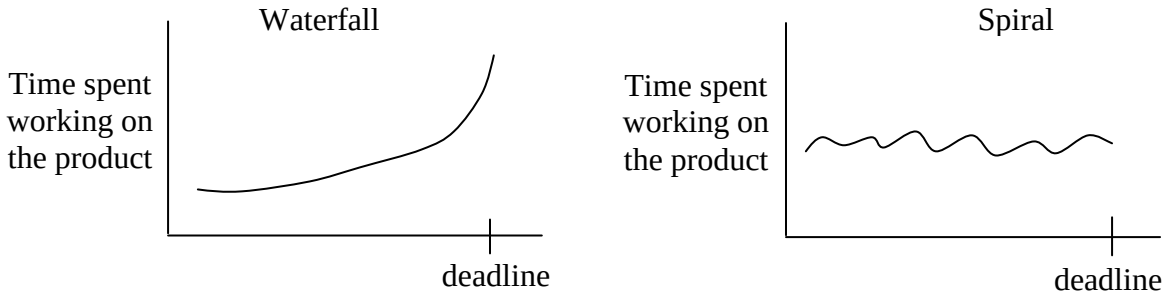
Question #	Points
1	(out of 3)
2	(out of 3)
3	(out of 3)
4	(out of 3)
5	(out of 3)
6	(out of 3)
7	(out of 4)
8	(out of 8)
9	(out of 5)
10	(out of 5)
11	(out of 3)
12	(out of 3)
13	(out of 3)
14	(out of 4)
15	(out of 3)
TOTAL:	(out of 56)

Instructions:

– Do not turn this page until instructed to do so.
– The exam is open book, open notes, closed laptops and other digital devices (to ensure fairness).
– You will have 60 minutes to work on the exam.
– There are 15 short-answer questions, intended to take you on average 4 minutes each. (Our answers to most questions are literally 2–3 lines.)
– Questions are worth between 3 and 8 points, for a total of 56 points.
– Do not spend too much time on any one question.
– If you need to refer to written sources (books, articles) in your answers, be sure to cite them.
– If you need additional space for any of the questions, use the last page. You may attach an additional sheet too, but be sure to put your name on it and to let us know that you have additional sheets.

– In your answers we will be looking primarily for solid understanding of the main concepts and ideas, and reasoning about how they interrelate, not evidence of strong memory or knowledge of "the right answer."

Q1 (3 pts): Consider the following two graphs that (roughly) represent the time spent developing a product at each point between the initial gathering of requirements (start) and the deployment / shipping (deadline). One graph relates to the waterfall lifecycle model, while the other corresponds to the spiral lifecycle model.



What causes the key differences, visible in the graphs, between the two models?

A: In the waterfall model it is not known until the very end how many hours the development effort will take, because (a) there is no feedback from customers after the requirements stage (and so the resulting "finished" product may be far from the actual customer needs, thus requiring further non-trivial work at the end), and (b) testing is done at the very end of the lifecycle, so problems that may significantly stretch the development are not caught early enough to allow for more realistic scheduling. As a result, projects on the waterfall model tend to require a lot of overtime and "death march"-style effort toward the end. In contrast, with the spiral model both customer feedback and testing are typically done as part of each iteration, thereby allowing for more accurate scheduling of the work.
(Note: This is a long and reasonably complete answer, intended to give you the full picture. We did not expect you to write this much in your answers – just to mark the important aspects.)

Q2 (3 pts): What is the software engineering term for what is described below? What is missing from it? (Note: You do not need to fill in the missing part – just state what it is.)

1. Create your list of DVDs online
2. We rush you DVDs from your list
3. Keep each DVD as long as you want
4. Return a movie to get a new one from your list

(Source: a recent ad by Netflix)

A: This is an example of a main success scenario (MSS) of a use case. Missing are any details of what might go wrong in one of the steps and how such a problem would be resolved.

Q3 (1+2 pts): Express in Big-O notation the number of communication paths in a team as a function of the team size N, if everyone needs to be able to communicate with:
(a) only the team manager?

A: O(N) – there's (N-1) people communicating with one, so (N-1) communication paths are needed.

(b) everyone else?

A: $O(N^2)$ – there's N people who need to each communicate with (N-1) of their colleagues, so there's N*(N-1)/2 communication paths. This accounts for the fact that the communication of person A to person B is the same as that of person B to person A.

In both cases, give brief explanations.

Q4 (3 pts): Identify at least five different stakeholders of a typical software industry project.

A: Developers, testers, project manager, marketing team, customers, shareholders, project owner, project champion, etc.

Q5 (1+2 pts):
(a) What are the main problems with doing too little high-level architectural design (a.k.a. "under-design") before coding begins?

A: With too little architectural design done upfront, the development team risks not considering (or even missing) an important component or interaction between components. This would later require expensive re-architecting of the product.

(b) What are the main problems with doing too much high-level architectural design (a.k.a. "over-design") before coding begins?

A: Doing too much of architectural design upfront may result in wasted effort, as some of the (especially more detailed) aspects may later need to change, for example, due to unforeseen technical difficulties. Also, an extra investment upfront in architectural design means that less time will be left in the project schedule for other important tasks that follow – detailed design, testing, coding…

Q6 (3 pts): From a design perspective, should a class expose a minimal but sufficient interface (i.e., set of public methods) to its clients or should it expose an interface that provides all methods that its clients might ever need? Explain.

A: Narrower interfaces (having fewer public methods) declare looser contracts between a class and its clients, which supports, rather than hurts, future extensibility. This is also related to the concept of information hiding.

Q7 (4 pts): Consider the definition of the method `installDoor()`, written in Java-like pseudo-code, from the `Contractor` class:

```
class Contractor {
   // any necessary instance variables
   // and methods defined here

   installDoor() {
      subcontractor = YellowPages.getSubcontractor();
      carpenter = subcontractor.getCarpenter();
      doorHandle = carpenter.getDoorHandle();
      doorBody = carpenter.getDoorBody();
      screws = carpenter.getScrews();
      door = carpenter.assemble(doorHandle, doorBody, screws);
      securityExpert = subcontractor.getSecurityExpert();
      securityExpert.installDoorSensors(door);
   }
}
```

What major design flaw is evident from the definition of the method? Which design principle(s) are violated? You can assume that any errors in method calls inside `installDoor()` are handled properly.

A: In the world of contractors and subcontractors, too much micromanaging occurs here. The method is tightly coupled to the classes of which `subcontractor`, `carpenter`, and `securityExpert` are instances.

Depending on details of the implementation of these classes makes it fragile, breaking the code if any of those details change. Violated principles include weak coupling, the Law of Demeter, and encapsulation.

Q8 (3+5 pts): Designing systems that can easily accommodate changes is an important part of what software engineers do. Imagine starting a design with a set of concrete (i.e., non-abstract) classes, but intending to eventually be able to make some of those classes abstract – as the need arises to refine (by sub-classing) the concepts that those initial classes represent. (For instance, the class `Animal`, initially concrete, may need to be refined into `Bird`, `LandAnimal`, and `MarineAnimal`, and so `Animal` will become an abstract class, holding only the common aspects of those three sub-classes.) Initially, however, you do not know what those refinements will be, so you can not predict which concrete classes will need to evolve into abstract ones.
(a) Would it make sense to start the design by making each one of your concrete classes derive from its own abstract class (with the hope that other sub-classes of those same abstract ones will become necessary)? Briefly explain.

A: Although possible, this approach will likely result in unnecessarily complex hierarchy of classes that the development team then will need to maintain in case some part of it ever becomes necessary.

(b) Assuming you do not want to take the approach in (a), how can you construct objects of your classes without having to make changes to their already existing clients?
Hint: The issue is that a client may be tempted to call a constructor of your (initially concrete) class, but eventually – as your class evolves into an abstract one – there will no longer be a constructor, and so the client code will break and be forced to change. Unless you design the construction process differently… How?

A: Have a public `getInstance()` method in your classes and make all constructors private. This way, the clients will have to call `getInstance()`, which will then invoke the proper constructor – either of the class if it is a concrete class, or of one of its sub-classes if the current one is abstract.

Q9 (2+3 pts):
(a) What changes to the design of your product would be required in order to deploy it in another country? (Internationalization is the technical term for this activity.) Be concise but specific.

A: The answer here would depend on the project and so it is necessarily open-ended. Typical considerations in internationalization are translating GUI items and documentation, as well as making the product culturally appropriate. The latter includes using colors properly (certain colors may be associated with different ideas in different countries); organizing presented information left to right versus right to left or top to bottom; incorporating ideas that are culturally popular in the target country; and changing or even removing certain aspects, depending on the legal and social environment of the target country.

(b) Focusing on just one type of changes (from those you mentioned in your answer above), briefly describe how specifically those changes might be accommodated in your current product.

**A: Consider the task of translation of GUI items. One option is for the text (or icon) associated with such an item to be dynamically retrieved from a database (based on a configuration of which country it is intended for) and placed visually on top of that GUI item whose other characteristics (size, color, etc.) are predetermined. There is a risk with this approach – in some languages (perhaps languages that are added later) the text may not fit the fixed size of the GUI item. A better option is to have all GUI items be pre-computed and stored in a folder (or database), along with all other items for the given target country. At runtime, each such item is retrieved (again, based on a configuration of which country it is intended for) and dynamically placed in the GUI. This approach avoids the fragility of the first option.
(Note: The above answer is longer that we expected from you on the exam.)**

Q10 (5 pts): Consider being hired to develop a software library for a customer. How do the various principles of *usability design* apply specifically to the design of the API? (Hint: One example of applying the principle of *mapping* is that function names in your library should clearly communicate the correspondence (i.e., the mapping) between the respective function and its operation.)

A: A well designed API should expose functions that enable clients to manipulate all necessary parts of the library, as well as functions to query persistent state. This is reflected in the principles of *visibility* and *feedback*. Further, the names of the library functions should *afford* their function. These names should also clearly communicate the *mapping* between each function and its particular operation, while hiding unnecessary complexity. Finally, the set of library function names should help the user of the API to easily develop an accurate *conceptual model* of the library's intent and operation.

Q11 (3 pts): Was your team's zero-feature release a horizontal or a vertical prototype? Was it an evolutionary or a throwaway prototype? Explain.

A: We expect that it was horizontal and evolutionary, in accordance with the idea of "flushing the tool chain" before having any features implemented. However, depending on your project risks and decisions, other combinations are possible. Note: Your justification is crucial here.

Q12 (3 pts): Name two risks that early prototyping helps to address.

A: Here are three risks (and there are more):
- reduced predictability about how long a project would take to complete;
- potential absence of interested customers;
- developing a product that is far from the customers' needs.

Q13 (3 pts): Why is it critically important for larger projects to use an automated build process?

A: The most important reason is that it produces consistent, reproducible build artifacts in a way that is less error-prone in comparison to doing so manually. Other correct but incomplete answers are that an automated build process saves development time and it also decreases the time to build the full product from the source code.

Q14 (4 pts): Consider the case of building a system that relies on third-party library code. Should these third-party elements be kept under version control (e.g., in CVS) along with the rest of your code? Why or why not?

A: It is important that third party library code be kept under version control as well. The reason is that the library's vendor may need to make changes to their code, moving to the next version, and if your code depends on aspects of a previous version of that library, this may unexpectedly break (and require reworking) some of your product.

Q15 (3 pts): Briefly explain the idea behind regression testing.

A: Regression testing is about running (accumulated) old tests along with the new ones, to ensure that new code (or newly introduced changes to old code) do not break code that used to work and that had previously successfully passed the old tests.

Note: Use this page as additional space (if you need it) and be sure to indicate which problem(s) your notes here correspond to.