Midterm Exam and Sample Solutions
CSE403 Summer 2005

July 25, 2005

Question #	Points
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	
11	
12	
13	
14	
TOTAL:	

NAME:_____

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 14 short-answer questions, intended to take you no more than 5 minutes each. (Our answers are literally 2–3 lines per question.)
– Each question is worth 3 points, for a total of 42 points.
– Do not spend too much time on any one question.
– 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. From a design perspective, would it be appropriate to implement a class X via inheritance from class Y in order to reuse the implementations of the many useful methods already defined in Y even if Y has other methods that have no meaning in the context of X? Explain briefly.

No, mere code reuse is not an appropriate use of inheritance. Inheritance defines an "is-a" relationship between types (classes), so every object of a derived sub-class is also a valid object of the corresponding super-class. Given that there are methods in Y that do not naturally belong in X, an object of type X would not be a valid object of type Y, so there is no "is-a" relationship. A more strict design criterion is the Liskov Substitution Principle, discussed in class; it would also be violated for the specific case of X and Y.

Q2. Is the following a reasonable expression for computing feature priorities?

```
Priority = (Value – Cost) / Risk
```

Explain briefly.

It depends. It prioritizes low-risk, profitable features, which for some projects could be a good thing. However, it negatively prioritizes high-value but high-cost or high-risk features, which could certainly be more important to implement first if we want to mitigate risks earlier or worry about more important features before going on to less valuable ones.

Q3. List one aspect in which software is different from other engineering disciplines and one aspect in which it is the same or similar. Explain briefly those differences and similarities.

Differences: shorter time to ship products; ability to change, patch, or extend at later time; easier to reproduce; no tangible artifact to accurately gauge progress; the underlying technology tends to change
Similarities: need planning to execute correctly, need a good foundation to assure high quality

Q4. To which type(s) of relationships is commonality and variability analysis (CVA) applicable?

(A) "`is-a`"
(B) "`has-a`"
(C) "`uses`"
(D) more than one of the above
(E) none of the above

Explain briefly.

CVA is applicable for "is-a" relationships. By looking for the common characteristics of objects (commonality analysis), we extract abstract categories which these objects share. Think of markers and pencils as both writing implements. So, Marker "is-a" WritingImplement and Pencil "is-a" WritingImplement. Variability analysis then defines how specific objects with the given commonalities differ, but all in the context of the established commonalities.

Q5. Use cases are one possible technique for specifying requirements that rests on describing the interactions between actors in the system. Are there situations in which use cases are impractical, even if it is clear what the interaction between the actors is? Explain and/or give an example.

The reason why use cases tend to be effective is that they can succinctly describe the key interactions in a system at a high level. This, however, assumes, at least, that the possible interactions are not too many and not overly complex. If a system inherently does not obey these two constraints, use cases will not be an effective tool for describing its requirements. One such example of a complex system is the game of chess.

Q6. Consider the following Java classes:

```
public abstract class Feline {
   private int age;
   private double weight;
   private double top_speed;

   public void run(double newSpeed) { ... }
   private void eat(double portionSize) { ... }
   // ... -> getXXX() and setXXX() methods here
}

public class Tiger extends Feline {
   private String breed;

   public Tiger(int age, double wt, double sp, String br) {
      this.breed = br;
      setAge(age);
      setWeight(wt);
      setTopSpeed(sp);
   }

   private void run(double newSpeed) { ... }
   public void camouflage() { ... }
   // ... -> getXXX() and setXXX() methods here
}
```

What is problematic with this code from a design standpoint? Can you relate it to a violation of one of the design principles. Explain briefly.

If Tiger extends Feline (as the code says), then it must be the case that all Tiger's are Feline's. However, the `run()` method in Tiger is private (i.e., invisible to the outside), whereas the `run()` method in Feline is public (i.e., visible to the outside). Since Tiger's do not have a public method `run()`, as Feline's do, they are not Feline's. This is a violation of the Liskov Substitution Principle (which says that all subtypes must be substitutable for their base types).

Q7. Give a basic reason for why "adding people to a late project makes it even later." (Fred Brooks)

Adding more people increases the communication overhead, which means that more time is taken communicating rather than making progress. The influx of people forces previously productive engineers into training, so they're no longer productive. When there are more people, especially new to the project, it becomes more likely that code changes will interfere with each other and create bugs.

Q8. What are the main risks of writing tests after coding is finished as opposed to writing them concurrently with writing the code? Explain briefly.

There are at least two drawbacks to writing tests after the code is finished.
One is that testing is a big unknown because it cannot be easily confined in the time it will take to be completed. Hence, leaving it for last runs the risk of shortchanging the quality of the product (if testing is done too hastily) or slipping the deadline (if it is done thoroughly), neither of which is an attractive proposition.
The second drawback is that when testing is done after coding, it loses one of its main benefits – that of serving as a working specification that the code must meet in order to be deemed correct. Supplying the tests later is more likely to mirror the code than to control it for correctness.

Q9. Imagine walking up to an unfamiliar computer monitor and wanting to turn it on. Then you notice that it has no physical buttons on the bezel (face), so you are stuck. Is this *primarily* an issue of

(A) lack of affordances,
(B) difference between the designer's and the user's conceptual models,
(C) lack of feedback, or
(D) strain on the cognitive load?

Explain briefly.

Affordances are cues to how to use a system. A button *affords* being pressed, and we are accustomed to them being on/off switches. The monitor has no immediately visible cues that we think will turn it on, so it is hard to figure out what to do.

Q10. Templates (or generics, in Java parlance) are used to generate a collection of classes when the type of the objects does not affect the behavior of the class's methods. Name a design technique that can be used when the type of objects does affect the behavior of the class's methods. Explain briefly.

Inheritance is such a design technique. Through inheritance subclasses can be defined that differ from the base class by supplying additional methods and/or by overriding existing methods from the base class. In both cases, the type of objects (instances of the subclass or of the base class) affects the behavior of the class's methods.
Note: Understanding templates is not at all necessary for answering this question. Templates are cited to abstractly highlight one of the essential characteristics of inheritance (by comparing it to that in templates).

Q11. Imagine having developed a set of unit tests but suddenly finding a bug that they do not catch. Naturally, you proceed to fix the bug. Why may you want to add to the set a new unit test that catches the bug, even though you have already fixed the bug?

The correct behavior becomes part of the specification since there is a unit test associated with it. Also, the bug could reappear later, especially if someone else were to modify the code, so a new unit test now can be used as a *regression test* later to catch that.

Q12. List one negative outcome that can happen if a project team does not confer regularly with the customer(s). Explain briefly.

The customer might reject the product because it differs wildly from their requirements. Gold-plating can occur where the customer sees an irrelevant feature that they paid for but didn't want. The product might be unusable because of poor UI. Without communicating with the customer, they can feel left out of the process and lose patience or faith in your company. Last but not least, the customer could change their mind at some point or evolve their requirements, but without having them in the loop, the team wouldn't know.

Q13. How does the number of communication paths depend (as a function) on the team size if everyone on the team needs to be able to communicate directly with everyone else? Explain briefly.

$O(n^2)$, where n is the team size (thus, polynomially). The number of pairs of people that need to communicate increases is "n choose 2".
Note: A lot of answers confuse "exponentially" with "polynomially." The former means $constant^n$, while the latter means $n^{constant}$ – a substantial difference from both practical and theoretical viewpoint.

Q14.
(A) Name one potential risk associated with the process of gathering requirements and one related to the process of designing software.

One risk with gathering requirements is that they are never final – customers often change their minds (or the nature of their business needs changes) after the initial requirements discussion. One risk with doing design is not communicating the structure of the design sufficiently well to its audience (of future developers on the team).

(B) List one or two practices that tend to be effective in mitigating or eliminating the specific risks you mentioned. Explain briefly.

Evolving requirements can be addressed through prototyping (as described in a handout) – when customers see something tangible, they know better if this is what they really meant and need. Designs are most effectively communicated when described using a set of standard notations (rather than home-grown ones that "look" understandable but only to the team that wrote them).

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