University of Washington
CSE 403 Software Engineering
Spring 2014

Final exam

June 6, 2014

Name: ***Solutions***

CSE Net ID (username): _____

UW Net ID (username): _____

This exam is closed book, closed notes. You have **50 minutes** to complete it. It contains 26 questions and 8 pages (including this one), totaling 100 points.

Before you start, please check your copy to make sure it is complete. Turn in all pages, together, when you are finished. **Write your initials on the top of *ALL* pages** (in case a page gets separated during test-taking or grading).

When you are asked for multiple answers, give answers that are as different as possible, and give the most important answers.

Please write neatly; we cannot give credit for what we cannot read.

Good luck!

Page	Max	Score
2	12	
3	20	
4	18	
5	15	
6	15	
7	12	
8	8	
Total	100	

1 True/False

(2 points each) Circle the correct answer. T is true, F is false.

1. [**T**] / **F** A refactoring is generally motivated by some specific task it will make easier, as opposed to simply improving the code.

   It's not cost-effective to refactor just to reduce entropy or satisfy your aesthetics. When a specific task (fix bug #2242, or add a particular feature) would be simplified if the code had a different structure, then it makes sense to first complete the refactoring, and after that to do the task, rather than to mix together the code restructuring and the task.

2 Multiple choice

(5 points each) Mark all of the following that can be true, by circling the appropriate letters.

2. The main goals of a code review are:

   (a) Is the code correct? ***This is a goal, but not a mail goal. Also, it's too hard to definitively answer in a code review.***

   [(b)] Is the code understandable?

   [(c)] Are the tests testing the right thing?

   (d) Producing new test cases

   (e) Producing fixes/patches for bugs

3. Circle all of the following that suggest a composition relationship and not aggregation (do not circle any that suggest both).

   (a) A "parent" element contains or owns zero or more "child" elements.

   [(b)] A strong life cycle dependency.

   (c) Shared possession.

   (d) At least one class in the relationship depends on the other (in the UML sense of dependency).

   (e) Both classes in the relationship depend on the other (in the UML sense of dependency).

3 Short answer

4. (4 points) Name two practices that increase the "bus number".

   (a) ***Code review***

   (b) ***Pair programming***

   (c) ***Complete documentation***

5. (6 points) Give an example of a problem for which Amazon found that a non-technical solution was much more effective than a technical solution.

   Problem: ***Filling boxes ("picking") was too slow.***

   Technical solution: ***Algorithms to optimize travel through the warehouse***

   Non-technical solution: ***Increasing font size so it was easier for people to read***

   Here is another example:

   Problem: **Inventory sometimes ran out in the warehouse even though not in the database, which permitted a customer to place an order that could not be filled.**

   Technical solution: **2-phase commit to prevent simultaneous ordering of the same (last) item by two customers**

   Non-technical solution: **Train the staff not to put items on the wrong shelf where they cannot be found during picking**

   Here is another example:

   Problem: **There are various policies and fees for small orders.**

   Technical solution: **Add code to implement these fees and policies on orders of various sizes.**

   Non-technical solution: **Re-evaluate requirements to discover that the fees were unnecessary anyway because no one was making purchases that small (to avoid the fees). Instead, disallow small orders.**

   It was not correct to say that efficient packing was a problem, complex spatial/weight packing algorithms were a technical solution, and letting humans pack was the non-technical solution. In this case, the algorithm was the effective solution, and the humans were present to account for deficiencies in the algorithm.

   This question is relevant to software engineering because it shows how software engineering is much more than just programming.

6. (6 points) Suppose that component A depends on component B. State Java code constructs that could cause this dependence. The answer should be English text, not code examples.

   (a) ***a method of A takes a B as a parameter or returns a B as a result***

   (b) ***code in A calls a method in B, or reads or writes a field of B***

   (c) ***A subtypes or subclasses B***

   (d) ***A has a field of type B***

7. (4 points) The dependency injection design pattern adds ("injects") a dependency. Describe, in one phrase each, where/when the dependency does not exist and where/when it does exist.

   Does not exist: ***at compile time***

   Does exist: ***at run time***

   A few answers that clearly described how dependency injection changes or moves dependencies in other ways were also accepted.

4 Design patterns

Give each answer in **one sentence** or less.

8. (4 points) Describe the most important difference between a library and a framework.

   A framework contains the main method, whereas a library does not. A library is code that you call, whereas a framework replaces main and calls your code, treating your code as a library.

   A common wrong answer is that a framework supplies the architecture whereas a library does not. This is incorrect for two reasons. First, the framework supplies **part** ***of the architecture, not the whole thing. Second, the library also supplies part of the architecture — just a different part.***

   Another common wrong answer is that a framework is bigger. That's usually true, but it's not necessary and is not a conceptual difference. There are similar problems with "a library solves a specific problem whereas a framework solves many problems" and "it is possible to reuse part of a library but impossible to reuse part of a framework".

9. (4 points) Give two disadvantages of the direct instantiation model that can be solved by using the factory method or factory class patterns.

   Consider the direct instantiation `new Date()`***.***

   (a) ***It creates a new thing, whereas you might want an existing one.***

   (b) ***It creates an object of a specified class (*`Date`*), whereas you might want a subclass.***

10. (6 points) Describe the two most important differences between composition and aggregation.

    The answers give details or consequences of the fact that the composite "owns" its elements whereas the aggregate does not.

    (a) ***We know from question 3 that there is a strong life-cycle dependency between the two objects, but you need to be more specific than that for full credit. If a composite is destroyed, so are its elements; but if an aggregate is destroyed, then its elements live on. Consider a theater: if it is destroyed, so is the box office (composition), but movies still exist (aggregation). It is*** **not** ***the case that if the elements are destroyed, the composite or aggregate is destroyed. Even for a composite, it could substitute a different element and continue to exist.***

    (b) ***In aggregation, but not composition, the aggregated ("contained") item can be shared by many aggregators.***

    (c) Composition requires at least one element object (the whole cannot exist without the parts), whereas aggregation requires ≥ 0 ***elements.***

    It is not true that composition requires exactly one subcomponent or subcomponents of specific types. It is not true that A vague answer like "composition is a stronger dependence and aggregation is a weaker dependence" or "a composite is wholly made up of its children" (or child, which is incorrect cardinality) does not earn credit, because we are looking for evidence of understanding rather than regurgitating text.

11. (4 points) A computer screen displays a set of nested elements (such as windows, panes, and buttons). Cocoa dispatches events from the inside out (visiting the smallest component first), whereas browsers dispatch events from the outside in (visiting the largest component first).

    State a design requirement that is convenient to implement in one of the models, but difficult or impossible in the other model.

    Model: ***Outside-in model***

    An outer component that must always run even if the inner component is also allowed to do some additional work. An outer component pre-empting an inner component, such as when an entire pane is grayed out or inactive.

    Another answer:

    Model: **Inside-out model**

    An inner component completely pre-empting an outer component.

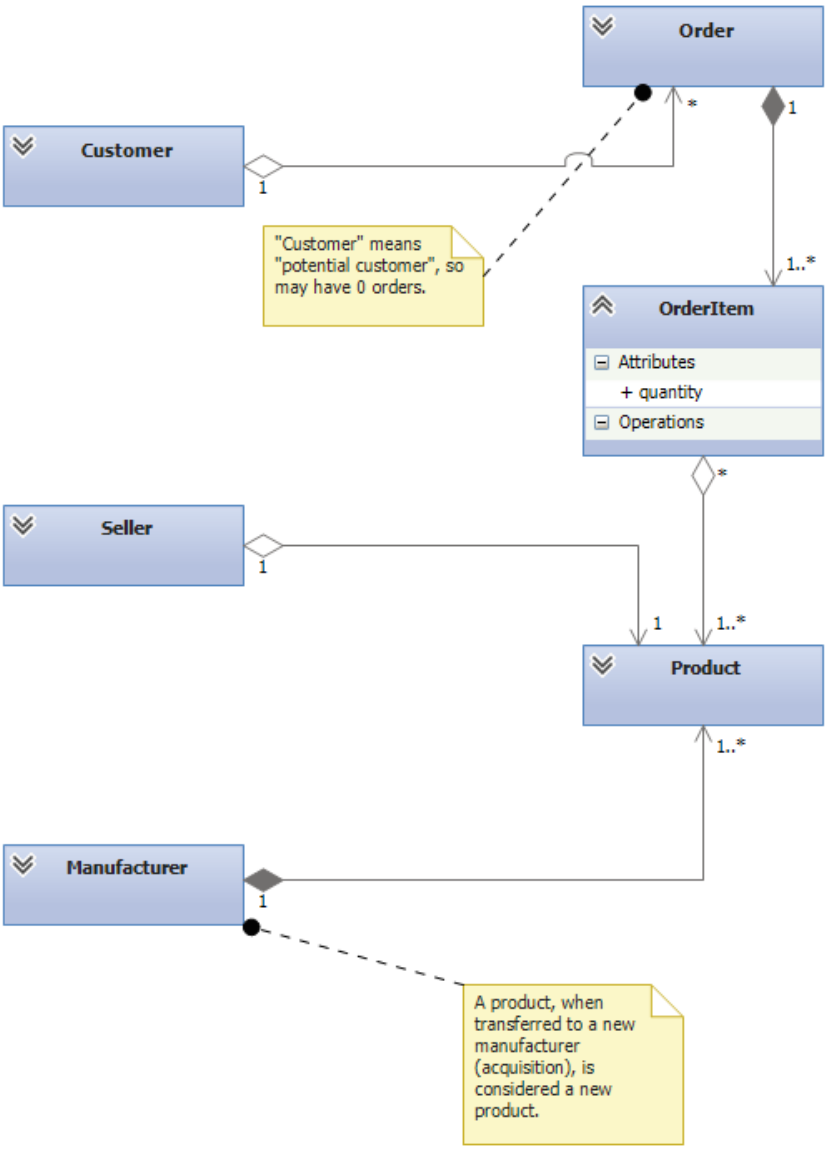
    Three other common types of answers were accepted: When an event that comes before the other in the dispatch order suppresses or overrides later events, when the results of handlers later in the dispatch order are dependent on the results of previous handlers, and if a global or local handler **always** ***handles the event and passing to other handlers is inefficient (this is a weaker answer).***

5 Design diagrams

12. (15 points) Draw the five most significant associations/dependencies (but no type relationships such as subtyping) between the classes shown. Do not add any new classes, and do not model anything other than associations. Add all multiplicities.

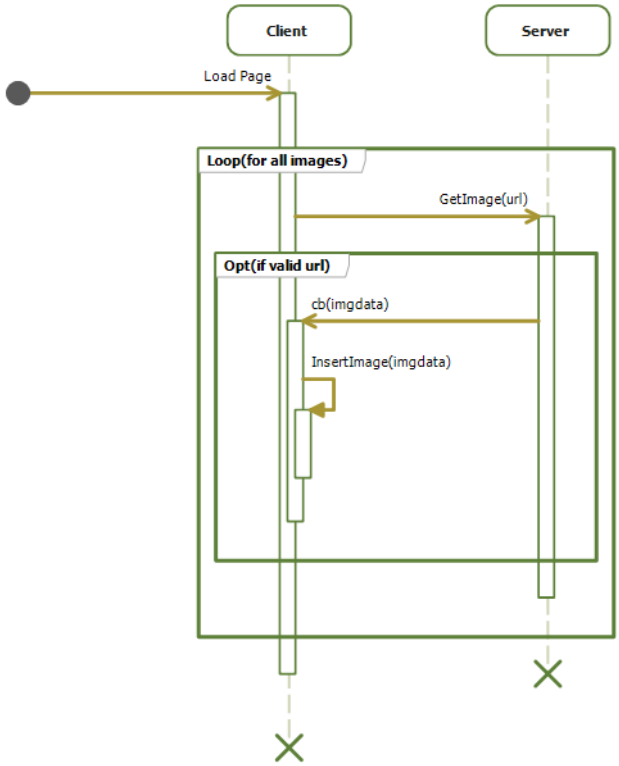
    You may add several words of description to a connector or a multiplicity if you feel it's necessary, but most full-credit solutions will not need any such description.

    Here is one possible solution:



We accepted many answers. Common mistakes were making no association between OrderItem and Product, multiplicities that seemed incompatible with the design of the system, and improper use of UML symbols.

(15 points) Consider the following sequence diagram:



13. **T** / **F** The arrow from client to server models a call that blocks and then returns. ***Async, so does not block and thus does not return. Calls a callback instead.***

14. **T** / **F** The arrow from server to client models a call that blocks and then returns. ***The call from server to client is an asynchronous callback.***

15. **T** / **F** The arrow from client to client models a call that blocks and then returns. ***This is a synchronous call.***

Is each of the following sequences a possible complete history of calls, immediately after Load Page? Calls to InsertImage are *not* modeled. Assume that the client and server are both infinitely multithreaded and that execution completes normally (without error).

16. **T** / **F** No calls. ***Maybe there are no images!***

17. **T** / **F** GetImage, GetImage, GetImage ***The IF statement need not execute if, for example, no URLs are valid.***

18. **T** / **F** GetImage, cb, GetImage, cb, cb ***cb is called at most once per GetImage call.***

19. **T** / **F** GetImage, GetImage, cb, cb, GetImage ***The last GetImage need not result in a call to cb.***

20. **T** / **F** cb, GetImage, cb ***The first call cannot be cb.***

21. **T** / **F** GetImage, GetImage, GetImage, cb, cb, cb ***All URLS are valid.***

6 Software engineering methodology

22. (4 points) In his guest lecture, Dennis Lee noted that once a project becomes late, it is likely to become even later. What was his main explanation for this?

    Once a project becomes late, the development team or their client thinks of the extra time as an opportunity to add features. But then the team is likely to miss the deadline for those new features, because the schedule slip was needed just to catch up.

    He explained this in terms of the Costco effect: if you can only shop once in a while (if your product will only ship once in a while), you want to fill your basket as full as possible (you want to cram in as many features as you can).

23. (4 points) If you discover a bug or other issue, you should fix it to improve your code quality. What are other engineering practices are essential to improving the code quality, typically *after* fixing the bug)?

    (a) ***Look for similar problems in other parts of your code.***

    (b) ***Perform a postmortem; change your process/practice to avoid that kind of mistake in the future.***

    (c) Write tests and/or add monitoring in case you do commit this or a similar error in the future.

    We did not accept "put the bug in the issue tracker, because not all bugs (for example, those found during development) need to be added to the issue tracker, and because that practice is not as important as the ones listed above.

24. (4 points) The primary purpose of code review is to improve the code (or design, or tests — whatever is being reviewed). State benefits of code review that do not improve such artifacts.

    Both answers are variants on "training the team".

    (a) ***Increases the bus number. Ensures that more people know the code. Teaches (new) employees about the abstractions, techniques, and patterns used by the system.***

    (b) ***Teaches every team member about design and programming practices and tricks they might not know.***

    "Improve the documentation" is not a correct answer. That's part of the code, or else is a separate thing that is being reviewed in its own right.

25. (4 points) State reasons that pair programming may deliver code with *more* functionality code than the same two people working independently.

    (a) ***Creativity: more ideas to choose from, more likely to choose a good one and not get stuck.***

    (b) ***Quick feedback, avoid poor design/implementation decisions. There are fewer bugs because two pairs of eyes are looking at and thinking about the same code. Catching bugs earlier is cheaper. Less need to go back and rework, which is slow and costly.***

    (c) The two employees can keep each other on-task. Taking turns gives each a break or change of pace without stopping work.

    Reducing the bus number, improving group knowledge, and training developers are benefits, but they are not relevant to the question unless you explicitly linked them to delivered code.

    It's not enough to say just better planning or two brains are better than one, without an explanation of how this affects delivered code. Merging and integration should not be a major time cost unless your project is very badly designed or modularized.

26. (4 points) State reasons that pair programming may deliver code with *less* functionality than the same two people working independently.

    (a) ***Some people don't work and/or communicate effectively with another person present. For instance, stopping to explain may interrupt flow.***

    (b) ***Need to explain/discuss tradeoffs and get consensus, preventing people from going at their own pace. They might even over-discuss issues that aren't all that important in the big picture.***

    (c) For a straightforward task that these two developers can do relatively easily (calling the developers "experienced" is a weak way of saying this), you don't need two people to mitigate risks in the design and coding, but pair programming suffers a keyboard bottleneck.

    (d) Two developers, not one, have to come up to speed on the problem and the codebase; the learning period is amortized over fewer developer-hours.

    We gave partial credit for "the code might be better and more concise" — that may sometimes be true, but not always, and it's not among the most important factors.

    Some people said that there will be more, and some said that there will be less, delivered code when one team member is inexperienced. Some of these answers were acceptable, but others were not. For example, answering an inexperienced developer's questions would have to be done whether or not the team is using pair programming.