University of Washington
CSE 403 Software Engineering
Spring 2011

Midterm exam

Friday, May 6, 2011

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 22 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.**

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

Good luck!

Page	Max	Score
2	16	
3	9	
4	17	
5	19	
6	18	
7	10	
8	11	
Total	100	

1 True/False

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

1. **T / $\boxed{\textbf{F}}$** Showing your customer a mockup of the UI is one good way to get feedback while gathering requirements.

2. **$\boxed{\textbf{T}}$ / F** For an announcement within your team, email is more appropriate than making the announcement during a meeting.
Meetings should be reserved for discussions — two-way communication — rather than announcements which are one-way communication.

3. **T / $\boxed{\textbf{F}}$** When evaluating a UI via paper prototyping, the materials should have a crisp, professional appearance, to aid in obtaining good feedback.

4. **$\boxed{\textbf{T}}$ / F** Cohesion refers to elements in the same module, whereas coupling refers to elements in different modules.

5. **$\boxed{\textbf{T}}$ / F** Suppose you are able to find the true revealing sub-domains for a software system, and you design unit tests for it accordingly. Later, if you change your implementation, it is also necessary to update your tests so that they still cover the revealing sub-domains, which may have changed.
A revealing subdomain corresponds to a sequence of choices made by the program, so it is a feature of the implementation rather than the specification. The existing unit tests would need to be supplemented to consider the revealing subdomains of the new implementation.
(In the handed-out version of the exam, this question was ambiguous, so we gave credit for both answers.)

6. **$\boxed{\textbf{T}}$ / F** When designing tests, if partitions are chosen perfectly, there is no point to testing boundary values near the edges of the partition.
Suppose your partitions are chosen perfectly. If the program behaves properly for any element in the whole partition, then it behaves properly for every element in the whole partition, including ones at the boundary. You don't have to test multiple boundary values — just test any one value.

7. **$\boxed{\textbf{T}}$ / F** Glass-box tests designed for one implementation of specification S are valid to use when testing another implementation of specification S.
"Glass-box" indicates the process by which the test inputs were chosen. No matter how a test's inputs are chosen, the test's inputs and oracle are still valid for every other implementation of the given specification.

8. **T / $\boxed{\textbf{F}}$** In an implementation of the singleton pattern, there is no constructor.
There is a constructor, but it is private.

2 Multiple choice

9. (3 points) Match each lifecycle model with its definition, by drawing a line connecting them.

(a) code-and-fix	(a) assess risks at each step; do most critical action first
(b) evolutionary prototyping	(b) build an initial small requirement spec, code it, then "evolve" the spec and code as needed
(c) spiral	(c) build initial requirement specs for several releases, then design-and-code each in sequence
(d) staged delivery	(d) standard phases (requirements, design, code, test) in order
(e) waterfall	(e) write some code, debug it, repeat (i.e. ad-hoc)

a-e; b-b; c-a; d-c; e-d

10. (3 points) A windowing system uses the notion of "damage" for what purpose?

    (a) to ensure correctness
    (b) to improve efficiency
    (c) all of the above
    (d) none of the above

    Damage is used to improve efficiency. It would be perfectly legal, but slow, to redraw the entire window or entire screen whenever any part of the display changes or becomes visible, but damage indicates exactly what parts need to be redrawn.

11. (3 points) Which of the following are appropriate for a requirements document? Circle all that apply.

    (a) Multiple users will be able to log on without experiencing conflicts or slow load times.
    (b) The program will not have any bugs.
    (c) If the systems detects a major issue, it will save state and restart.
    (d) The user will be able to modify the location of the program's save location on their machine.
    (e) The program will disconnect the user if they enter an invalid value in any section.

    A, C, D

3 Short answer

12. (4 points) The later a problem is found in the software development process, the harder it is to fix. Explain why, in one sentence.

    - ***Other work may be built on a flawed foundation, so correcting the original problem may require significant rework.***
    - ***If the software has already been widely distributed, it might be expensive to get a fix out.***
    - ***The developer(s) involved might have forgotten the context surrounding the bug, so it'll be harder to fix.***
    - ***The bug might be located in any existing code, and there's more to search through as time passes.***

13. (4 points) Give an example of a design pattern whose use is obvious from a class diagram but not from a sequence diagram. (Don't choose one that is built into (some) programming languages, such as inheritance.) Explain why, in 1 sentence.

    Composite: the members of a class are of a type that allows similar operations (perhaps they implement an interface in common with the container class). Observer: especially easy if there's an $\langle$observes$\rangle$ notation on an arrow.

    For many patterns it's possible to argue either way (and we were looking for your argument, not just a name). A common pitfall here was conflating class and object diagrams.

14. (4 points) Give an example of a design pattern whose use is obvious from a sequence diagram but not from a class diagram. (Don't choose one that is built into (some) programming languages, such as iteration.) Explain why, in 1 sentence.

    Factory: an actor creates an object in response to a call, and the caller subsequently sends messages to the newly created object. Decorator: every message to the decorator object is followed by a call to the object it decorates.

15. (5 points) Event-driven programming is appropriate for user interfaces and user interaction.

    - Give a *different* example domain.

      Networking. Disk access. Operating system interrupts. Web servers. A thermostat (takes action when the temperature is too high or low).
      Any user interface is a wrong answer: it is the same domain, not a different one.
    - In one phrase or sentence, state the characteristics of domains where event-driven programming is appropriate.

      Unpredictable delays.
      Other acceptable answers are: Unpredictable events. Real-time device control. Asynchronous control.
      "Non-sequential" is not an adequate answer by itself; it is not a necessary condition. "User input" is too restrictive.
      A surprising number of people gave characteristics that were violated by either user interfaces or their answer to the other part of the question; this inconsistency could have been an indication to them that their answer was not correct (usually, it was too specific).

16. (5 points) Consider two components A and B. Two software engineers, Laurel and Hardy, measure the dependences between A and B. Laurel uses these dependences when computing cohesion, and Hardy uses these dependences when computing coupling. Is this possible, if both engineers are performing a sensible and useful computation? In 1–2 sentences, explain why or why not.

    Yes. Laurel is considering a larger module C that contains both A and B as implementation details. Hardy is considering the implementation of C, and thinking of A and B as modules.

17. (6 points) In 1 sentence each, give two distinct reasons that you should not commit compiled code (such as `.o` or `.class` files) to a version control repository.

    - ***Merge conflicts cannot be resolved. Another way of saying the same thing is that binary files are not diffable (by the standard text-based diff algorithms).***
    - ***Repetition of information in source and binary forms violates the DRY (don't repeat yourself) principle.***
    - ***Binary files such as `.o` files are architecture-dependent and may not be useful to others.***
    - ***Binary files may contain information such as timestamps that is guaranteed to create a conflict even if generated from the same source code by others.***
    - ***Bloat in the VCS because differences are huge.***
    - ***Timestamps might not be preserved.***
    - ***If there is a check-in without compiling, then they can be inconsistent with the source code.***

18. (8 points) It is cheaper and faster to fix known bugs before you write new code. Why? In one phrase or sentence each, give three reasons. Give reasons that are as different from one another as possible.

    - ***You are familiar with the code now. A related reason is that the bug will be harder to find and fix later.***
    - ***Later code may depend on this code. A related reason is that a bug may reveal a fundamental problem.***
    - ***Leaving all bugs to the end will make it harder to understand and keep to the schedule, because it's hard to predict how long bug fixing will take.***
    - ***An overfull bug database is demoralizing and is likely to be ignored.***
    - ***You will be able to add tests for the bug once it's been fixed to avoid future issues.***
    - ***Avoid feature creep.***

19. (8 points) After you find a bug but before fixing it, you should create a test case for it. In one sentence each, give three reasons that this is a good idea. Give reasons that are as distinct as possible.

    - ***Ensures that your fix solves the problem. Don't add a test that succeeded to begin with! A related reason is to avoid writing a test for a bug that you fixed, but that isn't the problem indicated by the original bug fix.***
    - ***It helps you understand the bug and define the desired system behavior. ("It documents the bug" or "it informs others of the bug" is wrong, because it is the purpose of your bug tracking system to document your bugs. If you meant something different, such as the good answers listed here, then please be more specific.)***
    - ***It helps you know when you are done with bug fixing. A related reason is repeatability, and efficiency when debugging: the test is easy to run in an automated way to determine whether your fix works.***

    Here are some more answers we accepted, even though they are really just reasons to write a test at all, and not reasons to write the test **before** ***you fix the bug:***

    - ***Helps to populate test suite with good tests. The test case may reveal other problems also, that would make sense to fix at the same time.***
    - ***Protects against reversions that reintroduce bug. It happened at least once, and it might happen again.***

20. (10 points) Consider a wrapper whose implementation logs each call that occurs.

    In no more than 2 sentences each, explain when the wrapper should be considered a decorator (and why), and when that same wrapper should be considered a proxy (and why).

    - Decorator: ***A decorator has different functionality but the same interface as the delegate. If the wrapper's specification requires it to do the logging, then it should be considered a decorator.***
    - Proxy: ***A proxy has the same functionality and the same interface as the delegate. If the wrapper has a lenient specification that permits but does not require it to perform logging, then it should be considered a proxy.***

21. (10 points) Recall that the interning pattern guarantees that any two objects with the same abstract value are represented by just one concrete object. Answer each part in one sentence.

    (a) Give a usage pattern (or its characteristics) in which the interning pattern uses less memory, compared to not using it, and explain why.

    A compiler symbol table, in which most symbols are used multiple times, so eliminating duplication saves memory.

    (b) Give a usage pattern (or its characteristics) in which the interning pattern uses more memory, compared to not using it, and explain why.

    A situation in which most objects have different values, so the overhead of the hash table used by the interning implementation outweighs the reduction in memory used by duplicate objects.

    (c) Give a usage pattern (or its characteristics) in which the interning pattern uses less time, compared to not using it, and explain why. Ignore effects that are really due to memory use, such as faster allocation.

    Interning makes comparisons complete faster (the `equals` method always first checks object equality), so if there are many comparisons, the speedup outweighs the time cost of interning, which is a search for an equal object.

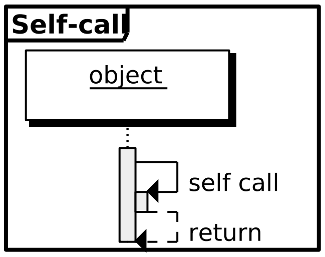
    (d) Give a usage pattern (or its characteristics) in which the interning pattern uses more time, compared to not using it, and explain why. Ignore effects that are really due to memory use, such as thrashing.

    If few equality checks are performed, then the speedup does not outweigh the time cost of performing interning. The main goal of interning is to save memory, so interning can be worthwhile even if it slows down the program.

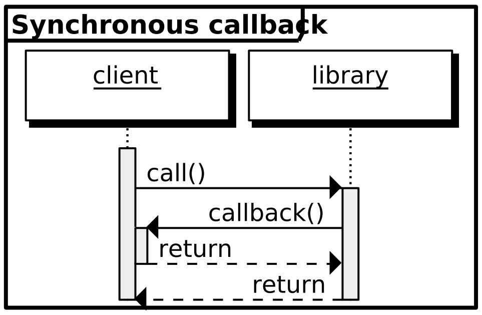
22. (11 points) Explain the difference between a self-call, a synchronous callback, and an asynchronous callback. Give a concrete example of each. Draw a sequence diagram for each, and explicitly mark the self-call or callback in each (one marked call in each).

    Several students answered the question as if it asked about calls rather than callbacks, and discussed issues such as blocking until a response is received. This is orthogonal to the notion of a callback. Also, some diagrams omitted return arrows, which is crucial to understanding the overall order of events, especially when distinguishing varieties of callbacks.

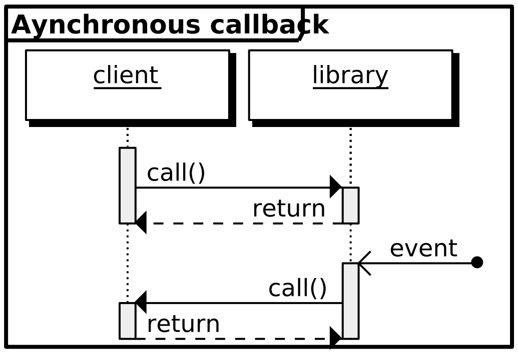
    Self-call: Explanation: ***A method call on `this`, or to the current module. It can be to any method — it doesn't have to be recursive. Recursion is an example, not a definition.***
    Example: ***Any call to a helper procedure.***
    Sequence diagram:



    Synchronous callback: Explanation: ***A client calls an external library that cannot do its work without delegating some work back to the client; the second call is the callback. Once the callback returns, the library eventually returns control to the client.***
    Example: ***A map's call to `hashCode`.***
    Sequence diagram:



    Asynchronous callback: Explanation: ***A client indicates to a library its interest in some event. The original call, that registers interest, returns immediately. At some indeterminate future time(s), the library calls the client to inform it that the event occurred.***
    Example: ***Event listeners and futures are the two main uses for asynchronous callbacks.***
    Sequence diagram:


Aynchronous callback
client
library
call()
return
event
call()
return