UNI:

Computer Science COMS W4156 Advanced Software Engineering Fall 2017 - Second Exam

December 7, 2017

Do not open the exam until the proctor tells you to do so. You may not use any books or notes. You may not use a calculator or any other device beyond a pen, pencil and eraser. Please write each answer in the corresponding space, continuing on the blank backs of pages if needed. Read through the entire exam before beginning to answer questions. Question 3 is long, with some intermediate pages to provide plenty of space for answers. It is not necessary to use all the space. The exam consists of 11 pages, with the last page saying only "(this page intentionally left blank)".

Name:

UNI (also put your UNI at the top of every page, since the pages will be separated during grading):

Problem No.	**Max Points**	**Points Scored**
1	10	
2	20	
3	30	
Total	60	

Problem 1 – Multiple Choice
(10 minutes, 10 questions, 1 point for each correct answer)

[1 point if correct, 0 points if incorrect or no answer or multiple answers or ambiguous]

Circle the letter that represents the **best** answer to each of the following questions.

1. What is the purpose of *mutation analysis*?
    a. Validate your design
    b. Validate the changes between sequential commits
    c. Validate your test suite ←
    d. All of the above
    e. None of the above

2. What should you *never* do before integrating third-party source code into your own project?
    a. Exploratory testing to determine the behavior of the parts you expect to use
    b. Apply tools that automatically generate class diagrams from existing code
    c. Push it into your version control repository
    d. All of the above
    e. None of the above ←

3. Which of the following is a *bug*?
    a. Software doesn't do something requirements say it should do
    b. Software does something requirements say it shouldn't do
    c. Software does something that requirements don't mention
    d. All of the above ←
    e. None of the above

4. Which of the following *best* describes regression testing?
    a. Zero bug bounce
    b. Test-to-pass
    c. Using a mocking framework
    d. All of the above
    e. None of the above ←

5. Which of the following might be *input* to a method invoked during a unit test?
    a. An object instance of that method's class
    b. Global data shared by all instances of the method's class
    c. Return values from external API calls made by the method
    d. All of the above ←
    e. None of the above

6. Which of the following are likely to be included among the *outputs* of a method invoked during a unit test?
    a. Sub-boundary conditions
    b. Teardown
    c. Parameters to external API calls made by the method ←
    d. All of the above
    e. None of the above

7. What is a reasonable set of boundaries for *range* equivalence classes?
    a. Root node, internal node, leaf node
    b. min-1, min, min+1, max-1, max, max+1 ←
    c. Data entered using a Dvorak keyboard
    d. All of the above
    e. None of the above

8. Which of the following should *never* be included in the issues of an issue tracker?
    a. References to specific commits in the version repository
    b. A specific sequence of steps to reproduce a bug
    c. Assignment to a specific developer
    d. All of the above
    e. None of the above ←

9. Which of the following should be done *before* a code inspection meeting?
    a. Inform the participants of which code will be inspected ←
    b. Delta debugging
    c. Minimize the truck factor
    d. All of the above
    e. None of the above

10. What is the *most significant difference* between acceptance testing and system testing?
    a. Acceptance testing validates the application in a customer's production environment whereas system testing is normally done prior to deployment ←
    b. System tests are normally executed from inside an interactive debugger whereas acceptance tests are usually run from the end-user's user interface
    c. System testing guarantees that a bug will never re-occur
    d. All of the above
    e. None of the above

Problem 2 – Vocabulary
(20 minutes, 5 questions, 4 points for each correct answer)

Explain the following terms in a few sentences and/or small drawings.
[0-4 points for each part, 0 for no answer or completely incorrect answer, note suggested points below add up to more than 4, but do not give more than 4 points. There may be other correct answers, or partial answers, beyond those listed: try to be consistent on partial credit for partially correct answers.]

1. User input validation

All user inputs should be checked before further processing. [2 points]

Validity vs. invalidity needs to be well-defined. [1 point]

Invalid inputs should result in an error message meaningful to a human end-user. [1 point]

Some invalid inputs can be sanitized and then processed without producing an error message. [1 point]

Any reasonable example of sanitizing user input, e.g., removing extra white space or replacing white space with an application-specific encoding. [1 point]

2. Branch coverage

Coverage refers to the percentage of the application code that is exercised by the test suite. [1 point]

Ideally, we want 100% coverage but this is rarely achieved in practice. [1 point]

Branch coverage refers to the test suite exercising both true and false cases for every conditional expression. [2 points]

Branch coverage may miss bugs in cases where conditionals include multiple subexpressions with AND/OR logical operators. [2 points]

3. Integration testing

Unit testing tests individual units whereas integration tests the interfaces between units. [1 point]

Integration tests is often achieved by replacing drivers and/or test doubles (or stubs, mocks, etc.) from unit testing with the corresponding program code. [2 points]

There are multiple strategies for integration testing, such as top-down vs. bottom-up. [1 point]

Ideally, integration tests should integrate only two units or previously tested integrations at a time. [2 points]

4. Greybox testing

Greybox testing checks intermediate products that are passed by the application to external systems, or between major subsystems within the application. [2 points]

Greybox testing checks external side-effects left over by the application. [1 point]

Greybox testing considers both the contents and formats of the external data produced. [1 point]

Any reasonable example of an external system where data passes from the application to that external system, e.g., from a web application to a browser client or to a database. [1 point]

Any reasonable example of side-effects from application execution, such as log files and temporary files. [1 point]

5. Test oracle

A test oracle is some mechanism for checking that the output produced for a given input is correct. [2 points]

Test oracles may be achieved by the human tester knowing what the correct answer should be for every input. [1 point]

Test oracles may be achieved by computing correct results some other way. [1 point]

Test oracles need to be able to check output correctness for every possible valid input. [1 point]

Test oracles may be implemented by assertions or by regular code. [1 point]

Problem 3 – Mini-Project
(30 minutes, 2 questions, 15 points each)

Recall the mini-project from the first exam. You are still working on software to manage task boards for agile teams. The primary features that the system needs to provide are:

- An administrator needs to be able to specify who are the members of each team and provide each team with a workspace.
- Team members should be able to make changes only in their own workspace.
- Team members need to be able to set up and manipulate task boards in their workspace.
- Task boards support To Do, In Progress, Completed and Overflow status categories.
- 

Task boards support Items, each of which consists of the description of a user story or another task to perform, a priority, a time estimate in "story points", who is assigned to do the task, and a status category (initialized as To Do).
- 

The Items of tasks boards automatically record when they move from To Do to In Progress and again when they move to Completed or Overflow, to track how much work time has elapsed between In Progress and Completed (or the time spent so far in the case of Overflow).

The actual questions to answer are on the following pages, parts A and B.

Part A – Unit Testing (15 minutes, 15 points)

You are a member of the development team for the task board system. Your pair has been assigned to implement the Item class. Your team uses test-driven development, which means you and your pair partner need to invent unit tests for the Item class *before* starting to code the class. Describe at least five TDD unit tests in prose or pseudo-code, and explain why you chose those test cases. Discuss any equivalence classes and boundaries that seem relevant to your test cases. Make any assumptions that seem reasonable about the other classes in the system, such as the Task Board class. Do not be concerned with how Items are stored persistently; a different pair is working on the database interface, for now your Items just need to exist as objects in memory. There is no single correct answer.

All test cases should be concerned with operations likely to be provided by the Items class. Do not give credit for any test cases that do not directly involve Items. Do not give credit for any test cases that involve checking a database, since the question says another team is doing that. Since these are supposed to be unit tests, do not give credit for any test cases that apparently would have to be executed from the application's user interface. Unit tests should always be automatable, so the test cases need to be implemented as code, with inputs supplied by code and outputs checked by code or assertions.

Note that throughout part A, "input" would probably but not necessarily be a parameter to some hypothetical method of the Item class, but could alternatively be global state accessed by a method in the Item class or data returned from a call made within the Item class to an API or other non-Item code. Analogously, "output" need not be a return value, could be changes to an object or other data that continues to be accessible after returning from the method, or parameters used in invoking an API or other non-Item code.

Are there at least N>=5 test cases left over after ignoring all those excluded above? Anything testing something about an Item's user story, priority, time estimate, personnel assignment, status category and/or elapsed time tracking is ok. They do not need to test all of these. [N points, up to N=5. If there are only two acceptable test cases, give 2 points. If there are 5 acceptable test cases, give 5 points. If there are 33 acceptable test cases, give 5 points.]

At least one test case that involves providing an input with an invalid value to some hypothetical method of the Item class, with an explanation as to why that is an invalid value. Also at least one test case that involves checking that the code under test responds with an error of some sort upon receiving an invalid value, with some explanation of the error response and how it is checked. Both parts might or might not be part of the same test case. [2 points for invalid input and 2 points for checking error response, total max 4 points.]

At least one test case that provides valid input described as a member of a specific equivalent class, which an explanation of that equivalence class. [2 points]

At least one test case that provides valid or invalid input described either as not a member of a specific equivalent class, or as a member of a second equivalence class that does not overlap with a first equivalence class, with an explanation of the difference. [2 points]

There should be at least one test case that provides input described with respect to a boundary of a specific equivalence class, with an explanation of that boundary (could be just below, at, or just above the boundary, any of these are ok). There are definitely equivalence classes with ranges, and thus at least lower boundaries, since story points should not be negative and time elapsed should not be negative. [2 points]

Continue your answer for part A on this page if necessary (you can also use the backs of pages).

Part B – System Testing (15 minutes, 15 points)

In this scenario, you are a member of a separate testing team, not the development team. The features sketched in the mini-project overview have all been completed by the development team. Your testing team's job now is to test the task board at the system level prior to the first demo for the customer. Make any assumptions that seem reasonable about the system's user interface. Describe at least five specific examples of system level tests, written as prose or pseudo-code, and explain why you chose those test cases. Discuss any equivalence classes and boundaries that seem relevant to your test cases. There is no single correct answer.

All system-level test cases should be concerned with activities that a human user could do from a hypothetical user interface. The test cases do not need to be automatable, but they should be repeatable, in the sense of instructions for a human tester to follow - both to provide user-level inputs and to check the correctness of user-level outputs (or lack of output, e.g., the application crashes or does nothing when it is supposed to do something). Ignore all test cases that do not fulfill these criteria. It does not matter whether the test cases seem to assume a web UI, a mobile UI, or a custom client UI.

Note that throughout part B, "input" is necessarily something the human can enter into the application, either directly with keyboard/mouse (e.g., numbers, strings, selections from a drop-down menu), indirectly via a file, or using some other mode of input (e.g., microphone, camera, touch gesture). "Output" should be something produced by the application that is directly or indirectly available to a human, e.g., updating the screen, making sounds, or writing to a user-level file or other external device. User-visible output could also include doing nothing or crashing (although the latter is never the desired outcome), but should not require the human user to inspect memory state using a debugger or other tool, access internal log files, intercept network I/O, etc.

Are there at least N>=5 test cases left over after ignoring those not accessible to an external human user? Anything testing something mentioned in the mini-project setup or assuming a reasonable extension to a fancier task board system is ok. [N points, up to N=5. If there are only two acceptable test cases, give 2 points. If there are 5 acceptable test cases, give 5 points. If there are 33 acceptable test cases, give 5 points.]

At least one test case that involves the human user providing an invalid input at the application UI level, with an explanation as to why that is an invalid value. Also at least one test case that involves the human user checking that the application under test responds with an appropriate error response upon receiving an invalid value, with some explanation of what error response should be expected and how it is checked or noticed by the human user. Both parts might or might not be part of the same test case. [2 points for invalid input and 2 points for checking error response, total max 4 points.]

At least one test case where the human provides valid input described as a member of a specific equivalent class, which an explanation of that equivalence class. [2 points]

At least one test case that provides valid or invalid input described either as not a member of a specific equivalent class, or as a member of a second equivalence class that does not overlap with a first equivalence class, with an explanation of the difference. [2 points]

There should be at least one test case that provides input described with respect to a boundary of a specific equivalence class, with an explanation of that boundary (could be just below, at, or just above the boundary, any of these are ok). There are definitely external equivalence classes with ranges, since at least story points should be user-visible. [2 points].

Continue your answer for part B on this page if necessary (you can also use the backs of pages).

(this page intentionally left blank)