# Grading Rubric for COMS W4156 Second Exam – December 6, 2016

Problem 1 – Multiple Choice (20 points). 2 points for each correct answer. -1 point for each wrong answer or if multiple answers are selected (or if there is any ambiguity as to which answer is selected). 0 for left blank.

1. e – none of the above
2. c - push it into your version control repository
3. d - all of the above
4. e – none of the above
5. a - non-developer testers usually perform system level testing rather than unit testing
6. a - a text box intended to allow only alphanumeric characters also allows the ';' character
7. c - exploratory testing is a term often used in software engineering whereas I made up the term extinction testing
8. b - invalid input should result in an error message understandable by the user
9. a - take notes about any problems found
10. d – all of the above

Problem 2 – Vocabulary (30 points). 0 points if blank or totally wrong. 3 points for a correct answer. Use your judgement in awarding partial credit, 1 or 2 points, for answers that aren't really correct but sort of vaguely have some clue, or that say something correct but also say something wrong; try to be consistent across students. Do not grade in half-points, i.e., no 0.5 or 2.5.

1. Test-driven development – A development technique where you first write a test that fails before you write new functional code that passes the test.
2. Branch coverage – A testing method that aims to ensure that each possible result (true and false) of a decision point is executed at least once. Or, alternatively, the testing metric that quantifies the percentage of branches that have been exercised.
3. Regression – When a new change to the software introduces bugs into older code, so test cases that previously passed now fail after the change. The regression test suite is typically the complete set of tests available.
4. Refactoring – Making a sequence of small changes to the code that alter the internal structure of the code without changing its external behavior or functionality. You need to retest after every small change to ensure that nothing has been broken.
5. Project Velocity – Computed by estimated days of work divided by number of actual days to get that work done, accounting for overhead in team development time. (It's ok if velocity is defined by an equation rather than prose, as long as the equation is correct.)
6. Iteration Review – Team meeting at the end of an iteration to discuss how to improve the software process. Also may compute metrics from the just-finished iteration, such as project velocity, coverage, and bug rate.
7. Spike – Block off a period of time, such as a week, and devote it to some activity where time estimates are needed, most notably bug-fixing. Use the number of bug-units fixed during this time to calculate time estimates for the remaining bug fixes.
8. Static whitebox testing – Automated tools that analyze the source code, and human code reviews that read the source code, in both cases without executing the code.
9. Dynamic whitebox testing – Executing the code with particular concern for which specific code is being executed, e.g., to determine coverage and try to force previously unexercised code.
10. Test-to-pass – Initial testing to check that the software works at all before trying to break (test-to-fail).

I cannot think of any reasons to draw pictures, but if students do draw pictures check that they make sense.

## Problem 3 – Mini-Project (50 points)

The question parts are open-ended, and there is no single “correct” answer. The points actually add up to 50 this time.

### Part A Code Inspection (20 points)

4: The codebase is intended to be too big for a single meeting, so the students should describe some plausible way to divide up into multiple meetings based on time, files, lines of code, or some programming unit such as package, module, class, etc.

4: The main roles are moderator, recorder and reader. Designating the author of the code as the reader, or designating that someone else other than the author must be the reader, is fine either way, but someone has to be the reader. It doesn’t matter who the students say is the moderator and recorder as long as these are different people and neither is the same person as the reader or the author.

4: The participants in the meeting need to be told the specific code that will be reviewed in the next upcoming meeting, so they can prepare by reading that code in advance and noting any concerns, comments or questions they might have. The code should also compile cleanly (it doesn’t matter whether the students say anything about using static analysis tools).

4: During the meeting the reader should go through the code line by line, perhaps reading each function all the way through or alternatively switching to a called function to read it and later returning to the calling function. The inspectors should discuss problems with the code, not with the coder, but should not try to actually solve any non-trivial problems during the meeting. The students should mention what kinds of problems to look for, anything reasonable is ok for this (e.g., deviations from coding conventions, code smells, useless or missing comments).

4: The author(s) should fix the problems after the meeting, which is then verified by the moderator. If the problems were severe, then another code inspection meeting should be held.

### Part B Testing (20 points)

If a student includes greybox testing for part B, ignore it for now and credit whatever is said about greybox testing to part C.

4: The team needs to devise at least two unit tests for every method (or function, procedure, subroutine, etc.), providing valid and invalid input parameters, respectively, and use a testing framework to run them. Its ok to ignore getters/setters.

4: The team needs to devise at least two tests, providing valid and invalid data, for each system level input point where data is received from outside the system. The students do not need to address whether the system testing is automated or manual.

4: Use equivalence partitions to devise the unit test and system test inputs.

4: When a partition is a numerical, character, or length/size range, there should be boundary condition tests at, just before, and just after each of the two boundaries. When the equivalence partition is a set, there should be tests of values inside and outside the set.

4: 2 points each for two plausible system level examples that test workspaces, task boards, items, or something else mentioned in the problem.

It’s not necessary to say anything about UI testing or integration testing.

Part C Greybox Testing (10 points)

If a student includes greybox testing for part B, ignore it when grading part B and credit whatever is said about greybox testing to part C.

5: The student says something that indicates he/she understands that greybox testing involves forcing the task board system to communicate with another system (here the version repository and issue tracker), and then examining the API calls and data transferred from the task board system to the other system.

3: The student says something that indicates that he/she understands that greybox testing also involves checking that the external API calls respond as expected.

2: I hope some students realize that the task board, which has multiple users, necessarily has both a client and a server so there are also interactions and data transfers between these subsystems that should be checked.

It is not necessary to discuss details about logs, files, databases, etc.