Second Individual Assessment

Published | Assign To | Edit | ⋮

This is an individual assignment. You are NOT permitted to post, discuss or work together on the assessment questions with other students in the class or with anyone else other than the teaching staff. Your answers can be as long or as short as you want, but a long meandering answer that incidentally mentions something relevant will receive less credit than a short high-quality answer. Submit your answers all in a single document in response to this assignment on courseworks. You can submit as many times as you want until the deadline. Submissions via email, edstem, or any other means besides courseworks will not be graded and will receive zero credit.

Academic Honesty policy: All submissions will be automatically analyzed by turnitin.com for plagiarism/collusion. Plagiarism or collusion will result in zero credit on the assessment and referral to the relevant disciplinary committee.

Late policy: Submissions posted one day late (which includes everything from 1 minute late to 23 hours 59 minutes late) will receive maximum 75% credit; two days late maximum 50% credit; three days late maximum 25% credit; courseworks will not accept submissions more than three days late.

Maximum total 100 points. There is no credit for part 0, but 0A is required in the sense that we will not grade the graded questions if part 0A is missing or incomplete and you will receive zero credit for the assessment. Part 0B is optional and will not affect your grade.

Ungraded questions: part 0A is required and part 0B is optional.

part 0A (required): You have 100 total points to allocate among the members of your team. **Assign some number of points, possibly 0, to each member of your team based on what you believe was that student's relative participation in the team project, and *explain your scores*.** For allocating points, consider level of effort and contributions for anything relevant to the team project. Note equal participation is not the same thing as equal lines of code produced. It is very important that you explain the scores you give to each student, including yourself: *If you do not provide meaningful explanations, then your response will be considered incomplete and you will receive 0 overall on the assessment.*

For example, in a team where all team members participated more or less equally, the scores would be: Sheldon 25, Leonard 25, Howard 25, Raj 25, and your explanation would describe who did which part(s) of the team project effort.

In a team with a disparity in contributions, however, the scores might be: Homer 40, Marge 25, Bart 20, Lisa 15, and the explanation might describe Homer as doing most of the work followed by a discussion of a little bit of what the others contributed. Scores like these would most typically be provided by Homer. Marge might instead give these scores: Homer 10, Marge 50, Bart 20, Lisa 20. This is why we need every team members' explanations of who did what, and who did not do what they were expected to do, to fairly credit differential grading.

part 0B (optional): This is your opportunity to provide feedback about the course separate from the SEAS evaluations, which in the instructor's opinion do not ask the right questions. Is there anything that you think should have been covered but wasn't? Is there anything that was covered that shouldn't have been (e.g., everyone already knows from their introductory CS courses)? Is there anything that should have been discussed more, discussed less, or presented in a different way? Is there anything else you would change about the course, including contents, teaching methods, delivery, assignments, anything else you think relevant? Please explain. Suggestions for next year's offering of the course will be greatly appreciated.

Graded questions: There are two graded questions, part 1 and part 2, each worth 50 points.

Part 1 (50 points): Identify a class in the main branch of your team's codebase that violates at least one of the SOLID principles. Refactor that class, using standard refactoring operations that preserve system semantics, so that it no longer violates that principle and you do not introduce any violations of SOLID principles into another part of the codebase (e.g., do not "fix" the problem by moving the whole thing to another class!).

Do not change anything in your team's shared github repository or in any other shared space that could become visible to your other team members! Instead copy the repository to your personal machine (this should be a *separate* copy from any work you may be doing intended to eventually commit to the shared repository). Perform the refactoring operations in that separate private copy. See **https://docs.google.com/document/d/1wbF7M0vxuJLe4NJ-rcjM5Q1QA7BgzN2R/edit** (https://docs.google.com/document/d/1wbF7M0vxuJLe4NJ-rcjM5Q1QA7BgzN2R/edit) . Please do each repo-changing question in a separate branch (1B, 2A, 2B).

Part 1A (10 points): State which specific principle is violated and explain in what way your chosen class violates that principle. Then explain why your revised version no longer violates that principle.

Part 1B (30 points): Give a link to a private repository with both "before" and "after" versions of any code you changed, and make sure your mentor has access and the README states what to find where. The README should also describe the exact sequence of refactoring operations you used to edit the original version to produce the revised version.

Part 1C (10 points): Explain how you know that your refactoring did not accidently change the semantics of your service.

Alternative for part 1A: In the unlikely case that none of your team's code violates any of the SOLID principles, instead pick some subset of the main branch of your team's codebase and refactor it using standard semantics-preserving refactoring operations to intentionally violate one of the principles. For 1A, State which specific principle is now violated in the revised version of at least one class in the chosen subset of your team's codebase. Explain in what way that class (or classes) now violates that principle and why the original version did not. 1B and 1C should be the same as above, except now the before version has no violations and the after version has at least one violation. Warning: If your mentor and/or other members of your team can find violations, then (obviously) your mentor will be very skeptical of claims that there aren't any.

Part 2 (50 points): Consider the entire set of automated test cases that are part of the main branch of your team's codebase. (Do not consider any manual tests.) Do not change the test code, the code under test, or anything else in your team's shared codebase or in any other shared space that could be visible to your other team members. See **https://docs.google.com/document/d/1wbF7M0vxuJLe4NJ-rcjM5Q1QA7BgzN2R/edit** (https://docs.google.com/document/d/1wbF7M0vxuJLe4NJ-rcjM5Q1QA7BgzN2R/edit) . Please do each repo-changing question in a separate branch (1B, 2A, 2B). Both test (re)ordering and mutation testing were covered in class, so questions about these from students who obviously did not attend/watch those two lectures will be ignored.

Part 2A (25 points): Find some reordering of the tests such that at least one test produces a different result, pass vs. fail. Tell us both the original ordering and the problem ordering, and explain specifically what happens differently to produce the different result. In the case that your test suite is so bad that pretty much any reordering produces different results, fix your test suite so this is no longer the case, and give your mentor access to your private "fixed" repository. In the unlikely case that you have tried or analyzed all possible reorderings and there isn't any reordering of the test suite that could produce a different result for any of your team's test cases, explain why not. Warning: Your mentor will be skeptical! (Construct and run candidate reorderings of the test suite using a private copy of your team's repository. It's ok to use a test reordering tool or do this manually.)

Part 2B (25 points): Using conventional mutation operators that change only one individual statement at a time and always produce compilable/runnable code, construct "mutants" of methods/functions in the main branch of your team's codebase until you find a mutant that your test suite fails to kill. Tell us all the mutants you tried. For the mutant(s) that your test suite could not kill, explain why not, and write at least one new test case that indeed kills that mutant. In the case that your test suite is so bad that pretty much any mutant survives, fix your test suite so this is no longer the case, and give your mentor access to your private "fixed" repository. In the unlikely case that it's not possible to construct an unkillable mutant, explain why not. Warning: Your mentor will be skeptical! (Create mutants and run tests using a private copy of your team's repository. It's ok to use a mutation testing tool or do this manually.)

This is the end of the assessment.

Points	100
Submitting	a file upload

File Types	pdf, doc, docx, and txt

Due	For	Available from	Until
Dec 9, 2022	Everyone	Dec 6, 2022 at 12:01am	Dec 19, 2022 at 11:59pm

Second Assessment Rubric

You've already rated students with this rubric. Any major changes could affect their assessment results.

Criteria	Ratings	Pts
1a I. Main question A. SOLID Principle (2 pts) - Principle is not violated: -2 pts - Principle is indeed violated B. Explanation (4 pts) - No explanation: -4 pts - Principle above was not violated but student answer demonstrates understanding: -2 pts - Principle was violated but student answer is not detailed enough: -2 pts - Correct explanation C. Revised version (4 pts) - No attempt: -4 pts - The described SOLID principle is still violated despite attempt: -2pts - The refactoring works but breaks another SOLID principle: -2 pts - No SOLID principle violated II. Alternative question A. Refactored branch (4 pts) - Refactored branch 404: -4 pts - Refactored branch ok, but does not showcase semantics-preserving operations: -2 pt B. Explanation of why revised version violates principles (3 pts) - Principle was not violated in the first place but student answer demonstrates understanding: -2 pts - Principle was violated and is now fixed but student answer is not detailed enough: -1 pts C. Explanation of why initial version did not violate any SOLID principle (3 pts) - Principle was violated in the first place but student answer demonstrates understanding: -2 pts - Principle was not violated, but student answer is not detailed enough: -1 pt	This area will be used by the assessor to leave comments related to this criterion.	10 pts
1b Separate branch exists for Part 1B (3 pts) At least 1 commit to branch (3 pts)	This area will be used by the assessor to leave comments related to this criterion.	30 pts

Criteria	Ratings	Pts
SOLID principle violation named (ex. somewhere in commit messages) or README (3 pts) README lists a sequence of refactoring operations (5 pts) Commit messages/changes are reflective of sequence of changes listed in the README (6 pts) Are the changes/refactoring operations appropriate for the SOLID principle violation listed? (10 pts) - Ex. for the Single Responsibility Principle: is the functionality broken up into multiple, smaller classes?		
1c Explains how they know their refactoring did not accidentally change the semantics of the service (-10) No response (-3) Does not mention unit/integration tests run to validate (-3) Does not mention system tests run to validate	This area will be used by the assessor to leave comments related to this criterion.	10 pts
2a [1st case - Identifies problem reordering] i) Identifies at least one test that changes its result due to a reordering in the test suite. [10] show quoted text> (-10) No response **show quoted text** ii) Explains in detail how the reordering affects the invariants causing the test to change its result. [11] show quoted text> (-11) No response **show quoted text** (iii) If the explanation in (ii) identifies/misses a major structural issue in the test file, then check the “fixed” repository. Otherwise, give the full score. [4] If the “fixed” repository is checked: Performs restructuring of the test suite in the private repo such that results are consistent between original and problem reordering. show quoted text> (-4) No response **show quoted text** [2nd case - student claims that a valid reordering could never break the test suite]	This area will be used by the assessor to leave comments related to this criterion.	25 pts

Criteria	Ratings	Pts
Explains in detail the design of the test suite that avoids the test suite to break. [18] show quoted text> (-18) No response show quoted text Explanation contains concrete examples of reordering to support the claim showcasing the result is immune to the reorderings. This includes test reports with random ordering enabled [7] show quoted text> (-7) No response / All examples are not relevant show quoted text Here, asking for concrete examples is fair, given the students were asked to "construct and run candidate reorderings of the test suite."		
2b [Case 1 – Found a mutant] Student explains which mutation operators they tried, and the mutations are reasonable one-statement mutations (eg. deleted a statement, changed a boolean expression with a literal, replaced an arithmetic operation with a different one, replaced a variable etc.) [10 pts.] The student identifies and fixes an unkilled mutation that is not caught by the test suite [15 pts.] Mutation is reasonable and does not get killed by the existing test suite [5 pts.] There is a coherent explanation for why this mutation is not caught by the current test suite [5 pts.] The student provides a fix to the test suite (either amending a current test or adding a new one) that kills the mutant [5 pts.] [Case 2 – No mutant is killed in the current codebase] The student first fixes the test suite so that not all mutants survive. [0 pts, but next part cannot be granted without this] Grade according to [Case 1] based on the fix [Case 3 – No mutant is unkillable by the test suite] Valid and thorough justification for why no mutant can survive the test suite [25 pts] – This would be extremely rare, so grader should investigate thoroughly before granting marks for this answer	This area will be used by the assessor to leave comments related to this criterion.	25 pts
		Total Points: 100