

CS 520

Homework 2

Mutation Testing and Real Faults

Due: **Tuesday, October 20, 2026, 9:00 AM EDT**. Students may work together on this assignment, but each student must submit their own assignment. If you work with others, you must clearly state the names of the students you worked with. Your responses to questions must be your own, but there is no penalty for working together. You are allowed to use AI tools to assist with this assignment (in fact, it is expected).

Late assignments will not be accepted without **prior** permission.

This assignment takes time. Do not wait until the last minute to start the assignment.

Overview and goal

Homework 2 and 3 will, together, replicate a research study looking into effectiveness of mutation testing.

This assignment teaches students to work with legacy code, replicate research studies, and evaluate test suite quality.

In this assignment, you will read a research paper and apply their methods to produce software testing artifacts.

Getting started

Download and read “Are Mutants a Valid Substitute for Real Faults in Software Testing?” by René Just, Darioush Jalali, Laura Inozemtseva, Michael D. Ernst, Reid Holmes, and Gordon Fraser. We will refer to this paper throughout this document as *Just et al.*

<https://people.cs.umass.edu/~brun/class/2026Fall/CS520/hw2/mutantsFSE14.pdf>

Your goal in reading is not to understand every little detail but to understand what questions researchers were asking, what was the design of the experiments they used to answer those questions, and what tools they used to answer those questions. **Focus on Sections 1 and 2.**

Just et al. develop a methodology for measuring whether mutants are representative of real-world bugs. You will use the same tools and experimental design to create the testing artifacts used in their experiments, but on a new program.

The software systems you’ll use

Just et al. use three systems you’ll need to use in your work:

1. Their evaluation takes place on 5 programs in the Defects4J benchmark. This benchmark consists of different software projects, each with isolated software bugs that were made in their development. Each bug is carefully isolated and prepared for researchers to use in their research. You will pick one of the projects not used in the original Just et al. paper and apply the Just et al. methodology to replicate their experiments on bugs from this project. How Just et al. used programs from the Defects4J benchmark is described in Sections 2.1, 2.2, and 2.3 of Just et al. The Defects4J benchmark is available here: <https://github.com/rjust/defects4j>

You will select a single project from Defects4J that is not already used in Just et al., and then select five (5) defects from that project to use in your replication. Some defects may be more difficult to use with the various tools you’ll need, so you may attempt more and select those that work for you.

2. EvoSuite is a free automated test generation tool. <https://www.evosuite.org>
You will use EvoSuite to generate tests for your subject program as described in Section 2.4 of Just et al. (They use other tools as well, but you can just use EvoSuite.)
3. The Major framework for mutation testing. <https://mutation-testing.org/>
You will use Major to generate mutants for your subject program as described in Section 2.5 of Just et al.

Your Tasks

1. Set up your tools (Defects4J, EvoSuite, Major). Getting these tools to run is part of the assignment. Use AI to help you get them running — these are well maintained and well documented tools.
2. Select five (5) defects from a Defects4J project not already used in the paper. (You may find some defects more difficult to use with EvoSuite and Major in the later steps and return here to select different ones.)
3. Use EvoSuite to generate 20 test suites for each of the 5 defects, as described in Section 2.4 of Just et al. Specifically:
 - Generate each suite for the *fixed* version of the defect.
 - Target only the classes modified by the bug fix.
 - Use a different random seed for each of the 20 suites, and record the seeds.
 - Remove tests that do not compile, tests that fail on the fixed version, and flaky tests. Following Just et al., consider a suite free of flaky tests once it passes 5 times in a row on the fixed version.
4. Use Major to generate mutants for each of the 5 defects, as described in Section 2.5 of Just et al. Specifically:
 - Mutate the *fixed* version of the defect.
 - Mutate only the classes modified by the bug fix.
 - Use the mutation operators described in Section 2.5 of Just et al.
 - Run mutation analysis once for each of your 20 EvoSuite suites (not the developer-written tests), so that you have a kill matrix for every generated suite. You will analyze these results in Homework 3.

Deliverables

You will submit, on [Canvas](#), a zip file containing:

1. Submit a README.txt or README.md or README.pdf answering the following questions:
 - (a) Your name, whom you worked with, and what AI tools you used.
 - (b) Describe, in 100 words or fewer, the purpose of mutation testing.
 - (c) In 100 words or fewer, what is the coupling effect, and why does mutation analysis depend on it?
 - (d) List which Defects4J project, and which 5 defects in that project you selected.

- (e) For each of your 5 defects, describe the bug and its fix in one or two sentences. Which of the paper's fault categories (Section 3.2) does each fit?
 - (f) Describe your environment and how to reproduce your results: the versions of Defects4J, Java, EvoSuite, and Major you used; the random seed of each EvoSuite suite; the mutation operators you used; and the exact commands you ran.
2. Submit the test suites you generated using EvoSuite, as `.java` source files.
 3. Submit Major's output for each of your 20 EvoSuite suites, for each defect: the mutant log, the kill matrix (which tests killed which mutants), and the summary. Defects4J's `mutation` command writes these to the working directory, so copy them out before the next run overwrites them.

Organize your zip file as follows, replacing `<Project>` and `<BugID>` with the Defects4J project name and bug ID (e.g., `Cli-5`):

```
lastname_firstname.zip
  README.md                (or README.txt or README.pdf)
  <Project>-<BugID>/        (one directory per defect, 5 in total)
    test_suites/
      suite01/              (EvoSuite .java files for suite 1)
      ...
      suite20/
    mutation/
      suite01/              (mutant log, kill matrix, summary for suite 1)
      ...
      suite20/
```

Do not include checked-out project directories, compiled `.class` files, or build directories.

Grading

The assignment is worth 100 points. **If your zip file is missing the README, the test suites, or the Major output, you will get a grade of 0 for the corresponding part(s).**

We expect to grade the assignment approximately with this rubric:

- 5pt:* The README contains your name, the names of the students you worked with (if any), and the AI tools you used.
- 10pt:* The description of the purpose of mutation testing is correct, is in your own words, and is 100 words or fewer.
- 5pt:* The description of the coupling effect is correct, is in your own words, and is 100 words or fewer.
- 5pt:* The README names a Defects4J project that is not one of the five used by Just et al. (Chart, Closure, Math, Time, Lang) and lists the five selected defect IDs.
- 20pt:* For each of the 5 defects:
 - 2pt:* The one-to-two-sentence description of the bug and its fix is accurate and consistent with the Defects4J fix for that defect.

- 2pt: The defect is assigned to a fault category from Section 3.2 of Just et al., and the choice is reasonable given the fix.
- 5pt: The README describes the environment (Defects4J, Java, EvoSuite, and Major versions), the seed of each EvoSuite suite, the mutation operators, and the commands used, in enough detail that we could reproduce the results.
- 25pt: For each of the 5 defects (5pt each), 20 EvoSuite test suites are submitted as `.java` source files. The suites were generated for the fixed version of the defect with distinct random seeds, target the classes modified by the bug fix, and compile and pass on the fixed version (failing, uncompileable, and flaky tests were removed, as described in Section 2.4 of Just et al.). The suites follow the required directory layout.
- 25pt: For each of the 5 defects (5pt each), Major's output is submitted for each of the 20 EvoSuite suites: the mutant log, the kill matrix, and the summary. Mutants were generated for the fixed version, only for the classes modified by the bug fix, using the mutation operators described in Section 2.5 of Just et al. The output follows the required directory layout, and the files are from distinct runs (not one run copied multiple times or overwritten).

Appendix: Technical Hints

Java version: Defects4J 3.x requires Java 11. Using a different Java version can cause unexpected test failures and bugs that do not reproduce. If you have several Java versions installed, make sure Java 11 is the one on your `PATH` and in `JAVA_HOME` when you run Defects4J.

Useful Defects4J commands: Run `defects4j` with no arguments to see the full list, and see the Defects4J documentation for the options of each command.

- `defects4j pids` lists the available projects; `defects4j bids -p <Project>` lists the bugs of a project; and `defects4j info -p <Project> -b <BugID>` describes a bug, including the classes modified by its fix.
- `defects4j checkout -p <Project> -v <BugID>f -w <dir>` checks out the *fixed* version of a bug (use `b` instead of `f` for the buggy version).
- `defects4j compile` and `defects4j test compile` compile and test the checked-out version.
- `defects4j mutation` runs Major on the checked-out version. Use its option for running a given test suite (rather than the developer-written tests) to run mutation analysis with your EvoSuite suites.
- `defects4j coverage` runs code coverage analysis. You do not need it for this assignment, but you will in Homework 3.

Generating tests: Defects4J includes scripts in `framework/bin` for generating test suites with EvoSuite (`gen_tests.pl`) and for removing failing and flaky tests from a generated suite (`fix_test_suite.pl`). Using these is much easier than running EvoSuite by hand, and by default they target the classes modified by the bug fix.

Mutation operators: Major names its operators with abbreviations. The four groups in Section 2.5 of Just et al. correspond roughly to LVR (replace constants); AOR, LOR, SOR, ORU, and ROR (replace operators); COR and ROR (modify branch conditions); and STD (delete statements). See the Major documentation to check which operators your setup uses.

Output files get overwritten: `defects4j mutation` writes its output files to the working directory and overwrites them on the next run. Copy them into your submission directory after every run. A short script that runs mutation analysis for each suite and copies the results into `<Project>-<BugID>/mutation/suiteNN/` will save you time and mistakes.