CS 520
Homework 3
Are Mutants a Valid Substitute for Real Faults

Due: **Tuesday, November 10, 2026, 9:00 AM EST**. 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

Recall that Homework 2 and 3 will, together, replicate a research study looking into effectiveness of mutation testing: "Are Mutants a Valid Substitute for Real Faults in Software Testing?" by Just et al.
https://people.cs.umass.edu/~brun/class/2026Fall/CS520/hw2/mutantsFSE14.pdf

Just et al. ask three research questions. **In this assignment, you will replicate only the third research question (RQ3):**

> **RQ3: Is mutant detection correlated with real fault detection?**

This question is described in Sections 2.6.1 and 2.6.4 of Just et al., and answered in Section 3.3. You will *not* replicate RQ1 or RQ2 (Sections 2.6.2, 2.6.3, 3.1, and 3.2), which use developer-written test suites.

This assignment teaches students to work with legacy code, replicate research studies, evaluate test suite quality, and draw (and refrain from drawing) conclusions from empirical data.

In this assignment, you will use the test suites and mutation analysis results you produced in Homework 2, collect the remaining data RQ3 needs, apply the statistical analyses Just et al. used, and compare your results to theirs.

Getting started

Reread Sections 2.4, 2.5, 2.6.1, 2.6.4, and 3.3 of Just et al. Make sure you understand what Figure 4, Figure 8, and Table 5 show, because you will produce your own versions of them.

You will build on your Homework 2 work: the same Defects4J project, the same 5 bugs, the same 20 EvoSuite test suites per bug, and the Major mutation analysis results for each suite. If you find problems with your Homework 2 work (for example, suites that still contain failing tests), fix them, and describe what you changed in your README.

The Major output files. RQ3 needs to know, for each suite, which mutants the suite *covered*, not just which mutants it killed (see Task 4 below). If you did not keep the Major output that records this information in Homework 2, re-run mutation analysis for your suites and keep it this time. See the Technical Hints appendix.

Terminology

We use the following terms, following Just et al. (Section 2.4 and 2.6). All of them are defined separately for each bug.

- A test suite *detects the real fault* if it passes on the fixed version of the bug and fails on the buggy version *because of the fault*. (Failures because of compilation errors, configuration problems, or flakiness do not count.)
- $\tilde{T}_{fail}$ denotes a generated suite that detects the real fault, and $\tilde{T}_{pass}$ denotes a generated suite that does not.
- The *real fault detection* of a suite is 1 if the suite is a $\tilde{T}_{fail}$ and 0 if it is a $\tilde{T}_{pass}$. Because each suite either detects the fault or does not, this is a dichotomous (binary) variable.
- The *statement coverage* of a suite is the fraction of statements in the classes modified by the bug fix that the suite executes on the fixed version.
- The *mutation score* of a suite is the fraction of all mutants (generated for the classes modified by the bug fix) that the suite kills.
- For a pair of suites $\langle \tilde{T}_{pass}, \tilde{T}_{fail} \rangle$, the *coverage-controlled mutation score* of each suite is the fraction of mutants in M that the suite kills, where M is the set of mutants covered by *both* suites in the pair. A pair for which M is empty is discarded (Section 2.6.1 of Just et al.).

Your Tasks

1. **Determine real fault detection.** For each of the 5 bugs, run each of your 20 suites on the *buggy* version of the bug. Classify each suite as a $\tilde{T}_{fail}$, a $\tilde{T}_{pass}$, or *broken*:
   - If the suite fails on the buggy version, inspect the failing tests and confirm that they fail because of the real fault (for example, an assertion about the behavior the fix changed fails, or an exception is thrown in the code the fix changed). If so, the suite is a $\tilde{T}_{fail}$.
   - If the suite passes on the buggy version, it is a $\tilde{T}_{pass}$.
   - If the suite does not compile against the buggy version, or its tests fail for reasons unrelated to the fault (Just et al. verified this too; see the end of Section 2.4), the suite is *broken*. Exclude broken suites from the rest of the analysis, and report them.
2. **Determine which bugs can be analyzed.** RQ3 can be answered for a bug only if at least one of its suites is a $\tilde{T}_{fail}$ and at least one is a $\tilde{T}_{pass}$. (Just et al. applied the same restriction; see Section 2.6.4.) It is fine if some, or even all, of your bugs cannot be analyzed: **do not generate additional test suites.** Instead, report this, and explain it as described in the Deliverables. Tasks 5 and 6 apply only to the bugs that can be analyzed.
3. **Measure statement coverage.** For each bug, measure the statement coverage of each of your non-broken suites on the *fixed* version, over the classes modified by the bug fix.
4. **Compute mutation scores.** For each bug, compute the mutation score of each of your non-broken suites from your Major results (all mutants, coverage not controlled). Also extract, for each suite, the set of mutants it covered and the set of mutants it killed; you will need them for the coverage-controlled analysis in Task 6.
5. **Correlation analysis (Figure 8 of Just et al.).** For each bug that can be analyzed:
   - Compute the point-biserial correlation coefficient, and its p value, between mutation score and real fault detection.

- Compute the rank-biserial correlation coefficient, and its p value, between mutation score and real fault detection.
- Compute the same two correlation coefficients, with p values, between statement coverage and real fault detection.

Then, if at least two bugs can be analyzed, use the Wilcoxon signed-rank test, across those bugs, to test whether the correlation coefficients for mutation score are higher than those for statement coverage (as in the caption of Figure 8). Report the p value. If fewer than two bugs can be analyzed, explain why this test cannot be run.

6. **Comparing $\tilde{T}_{fail}$ and $\tilde{T}_{pass}$ (Table 5 of Just et al.).** For each bug that can be analyzed, form all pairs $\langle \tilde{T}_{pass}, \tilde{T}_{fail} \rangle$ of that bug's suites. (A bug with 5 $\tilde{T}_{fail}$ and 15 $\tilde{T}_{pass}$ suites has 75 pairs.) Then, twice — once with coverage *not controlled* (each suite's mutation score), and once with coverage *controlled* (each suite's coverage-controlled mutation score for that pair, discarding pairs with no commonly covered mutants):

   - Use the one-sided Wilcoxon signed-rank test over the pairs to test whether the mutation score of the $\tilde{T}_{fail}$ is higher than that of the $\tilde{T}_{pass}$. Report the p value.
   - Compute the $\hat{A}_{12}$ effect size: the fraction of pairs in which the $\tilde{T}_{fail}$ has a higher mutation score than the $\tilde{T}_{pass}$, counting ties as one half.
   - Report the number of pairs used (and, for the controlled analysis, the number discarded).

About statistical significance: Just et al. studied 357 real faults and 35,141 test suites. You are studying 5 bugs and 100 test suites. Many of your p values may be high (not statistically significant). **That is OK**: report the p values you get. The point of the assignment is to apply the methodology correctly and interpret its results honestly, not to get significant results.

Deliverables

You will submit, on Canvas, a zip file containing:

1. Submit a README.md or README.pdf (we recommend PDF, since it must include tables and a figure) answering the following questions:

   (a) Your name, whom you worked with, and what AI tools you used.

   (b) In 100 words or fewer, in your own words, what question does RQ3 ask, and why does the answer matter to people who use mutation analysis?

   (c) In 100 words or fewer, why did Just et al. control for code coverage? Explain, in your own words, what could go wrong if they hadn't.

   (d) In 100 words or fewer, why can RQ3 not be answered for a bug when all of its suites detect the real fault, or when none of them do?

   (e) Describe your environment and how to reproduce your results: the versions of Defects4J, Java, EvoSuite, Major, and any statistics tools (e.g., Python and SciPy, or R) you used; the exact commands you ran; and anything you changed or re-ran from Homework 2, and why.

   (f) For each of the 5 bugs, a table listing, for each of the 20 suites: its seed, whether it is a $\tilde{T}_{fail}$, $\tilde{T}_{pass}$, or broken, its statement coverage, and its mutation score. (This is the same data as in `results.csv`, below.)

(g) For each of the 5 bugs, one or two sentences on how you confirmed that the $\tilde{T}_{fail}$ suites fail because of the real fault (if there are any), and why each broken suite (if any) is broken.

(h) For each bug that cannot be analyzed, one or two sentences on why you think its 20 suites all detected the fault, or all missed it. (Think about how EvoSuite generates tests and their assertions, and about the bug and its fix.)

(i) A figure analogous to Figure 4 of Just et al.: for each bug that can be analyzed, the statement coverage and mutation score of its suites, grouped by real fault detection (0 or 1).

(j) A table analogous to Figure 8 of Just et al.: for each bug that can be analyzed, the point-biserial and rank-biserial correlation coefficients (with p values) for mutation score and for statement coverage. Also report the p value of the Wilcoxon signed-rank test comparing the two (or explain why it cannot be computed).

(k) A table analogous to Table 5 of Just et al.: for each bug that can be analyzed, with coverage not controlled and with coverage controlled, the Wilcoxon signed-rank p value, the $\hat{A}_{12}$ effect size, and the number of pairs used (and discarded).

(l) **Discussion.** In 300 words or fewer: Does your replication agree with the findings of Just et al. for RQ3? Support your answer with your results. Your replication is much smaller than the original study, so it is entirely acceptable to conclude that you have insufficient data to agree or disagree, as long as you explain why your data are insufficient. Also describe at least two threats to the validity of your replication, in particular ways in which it differs from the original study.

2. Submit all scripts you wrote to run the tools and to compute your results (e.g., shell, Python, or R scripts).

3. For each bug, submit a `results.csv` file with one row per suite (20 rows) and the following header:

   ```
   suite,seed,detection,statement_coverage,
       mutants_generated,mutants_covered,mutants_killed,mutation_score
   ```

   (on one line), where `detection` is `1` ($\tilde{T}_{fail}$), `0` ($\tilde{T}_{pass}$), or `broken`, and the remaining columns are empty for broken suites.

4. For each bug, submit the raw output of running each suite on the buggy version (at least the list of failing tests, with their failure messages and stack traces) and the raw coverage output for each suite.

5. If you re-ran mutation analysis, submit the Major output (as in Homework 2) for each suite you re-ran. If you changed or regenerated any test suites, submit them (as in Homework 2).

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.pdf)
  scripts/                  (all scripts you used)
  <Project>-<BugID>/        (one directory per bug, 5 in total)
    results.csv
    detection/
      suite01/              (output of running suite 1 on the buggy version)
      ...
```

```
    suite20/
  coverage/
    suite01/              (coverage output for suite 1)
    ...
    suite20/
  mutation/               (only if you re-ran Major)
    suite01/
    ...
  test_suites/            (only if you changed or regenerated suites)
    suite01/
    ...
```

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 scripts, the `results.csv` files, or the raw detection and coverage output, you will get a grade of 0 for the corresponding part(s).**

Bugs that cannot be analyzed. You will not lose points because some, or even all, of your bugs cannot be analyzed (all of their suites detect the real fault, or none do). The last three analysis items of the rubric below (the figure, the correlation analysis, and the comparison of $\tilde{T}_{fail}$ and $\tilde{T}_{pass}$) are divided equally among your 5 bugs. For a bug that can be analyzed, its share is graded on the analysis. For a bug that cannot be analyzed, its share is instead graded on whether you correctly identified that the bug cannot be analyzed (consistent with your raw detection output) and correctly explained why RQ3 cannot be answered for it. For example, if 2 of your 5 bugs cannot be analyzed, 2/5 of each of those items' points are graded on your explanation for those 2 bugs, and 3/5 on your analysis of the other 3 bugs.

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.

5pt: The description of RQ3 and why it matters is correct, is in your own words, and is 100 words or fewer.

5pt: The explanation of why Just et al. controlled for code coverage is correct, is in your own words, and is 100 words or fewer.

5pt: The explanation of why RQ3 cannot be answered for a bug whose suites all, or none, detect the real fault is correct and 100 words or fewer.

5pt: The README describes the environment, commands, and changes from Homework 2 in enough detail that we could reproduce the results, and the submitted scripts are the ones used.

15pt: For each of the 5 bugs (3pt each), every suite is correctly classified as $\tilde{T}_{fail}$, $\tilde{T}_{pass}$, or broken, consistent with the submitted raw output; the README explains how $\tilde{T}_{fail}$ suites were confirmed to fail because of the real fault, and why any broken suites are broken; and, if the bug cannot be analyzed, the README gives a reasonable explanation of why its suites all detected, or all missed, the fault.

10pt: For each of the 5 bugs (2pt each), `results.csv` is complete, and its statement coverage and mutation scores are consistent with the submitted raw coverage output and the Major output (from this assignment or Homework 2), and were measured over the classes modified by the bug fix, on the fixed version.

5pt: For each of the 5 bugs (1pt each), the figure shows statement coverage and mutation scores grouped by real fault detection or, if the bug cannot be analyzed, the README correctly explains why (see "Bugs that cannot be analyzed" above).

15pt: For each of the 5 bugs (3pt each), the correlation analysis (Task 5) is correct: the point-biserial and rank-biserial coefficients and p values for mutation score and statement coverage or, if the bug cannot be analyzed, the README correctly explains why. The Wilcoxon signed-rank comparison across bugs (or a correct explanation of why it cannot be computed) is part of every bug's share.

15pt: For each of the 5 bugs (3pt each), the comparison of $\tilde{T}_{fail}$ and $\tilde{T}_{pass}$ (Task 6) is correct or, if the bug cannot be analyzed, the README correctly explains why. For a bug that can be analyzed: the pairs are formed correctly; the coverage-controlled scores use only the mutants covered by both suites in a pair, and pairs with no commonly covered mutants are discarded; and the Wilcoxon signed-rank p values, $\hat{A}_{12}$ effect sizes, and pair counts are reported for both the controlled and the not-controlled analyses.

15pt: The discussion is 300 words or fewer, and its conclusion (agree, disagree, or insufficient data) is supported by the results and accounts for the scale of the replication. It describes at least two reasonable threats to validity.

Appendix: Technical Hints

Java version: As in Homework 2, Defects4J 3.x requires Java 11. Make sure Java 11 is the one on your `PATH` and in `JAVA_HOME` when you run Defects4J.

Test suite archives: The `-s` option of `defects4j test`, `coverage`, and `mutation` takes a test suite archive (`.tar.bz2`) like the ones `gen_tests.pl` produced in Homework 2. The archive's file name must have the form `<Project>-<BugID>f-evosuite.<N>.tar.bz2` (e.g., `Codec-3f-evosuite.1.tar.bz2`), and the archive must contain the test sources in their package directories (e.g., `org/apache/commons/codec/language/DoubleMetaphone_ESTest.java` and the matching `_scaffolding.java` file). If you kept only the `.java` files from Homework 2, re-create the archives with `tar -cjf` from the directory that contains the top-level package directory (`org/`).

Running a suite on the buggy version: Check out the buggy version with `defects4j checkout -p <Project> -v <BugID>b -w <dir>` and run a suite on it with `defects4j test -s <suite archive> -w <dir>`. It prints the number of failing tests and writes the failing tests, with their stack traces, to the file `failing_tests` in the working directory (the file is empty if all tests pass); copy it out after every run.

Defects4J also includes the script `run_bug_detection.pl` (in `framework/bin`), which runs every suite archive in a directory (`-d`) on the buggy version. It writes a CSV file `bug_detection` to its output directory (`-o`) that labels each suite `Fail` (at least one test failed), `Pass`, or `Broken` (the suite could not be run), and it keeps a log for each suite in `bug_detection_log/`. Note that `Fail` only means that some test failed, not that it failed *because of the fault*: you still have to inspect the failures (Task 1). The script also skips suites that already have a row in the output file, so use a fresh output directory if you re-run it.

Measuring statement coverage: Run `defects4j coverage -s <suite archive> -w <dir>` on a checkout of the *fixed* version. By default, it instruments only the classes modified by the bug fix. It prints line and condition coverage; use line coverage as statement coverage (Just et al. used Cobertura, which Defects4J also uses). It writes its detailed results to `coverage.xml` in the working directory; the `line-rate` attribute of the top-level `<coverage>` element (equivalently, `lines-covered` divided by `lines-valid`) is the line coverage. Copy `coverage.xml` out after every run.

Mutants covered and killed: `defects4j mutation` writes `summary.csv`, `kill.csv`, and `mutants.log` to the working directory. `kill.csv` (which is the "kill matrix" from Homework 2) has one row per mutant: the mutant's number (as in `mutants.log`) and its outcome, which is one of `FAIL`, `TIME`, `EXC`, `LIVE`, or

UNCOV. FAIL, TIME, and EXC mean the suite killed the mutant (a test failed, timed out, or threw an exception); LIVE means the suite covered the mutant but did not kill it; and UNCOV means the suite never executed the mutated code. So a suite's covered mutants are all mutants that are not UNCOV. Check your counts against MutantsKilled and MutantsCovered in summary.csv. Because all your runs for a bug mutate the same fixed version, the mutant numbers in kill.csv refer to the same mutants for every suite of that bug; you can confirm this by checking that the mutants.log files are identical. If your Homework 2 output does not include kill.csv for every suite, re-run mutation analysis and keep it.

Which mutation score: defects4j mutation prints the mutation score as two percentages, e.g., Mutation score: 58.0% (56.8%). The first is killed mutants divided by *covered* mutants; the second, in parentheses, is killed mutants divided by *all generated* mutants. This assignment uses the second (MutantsKilled divided by MutantsGenerated in summary.csv; see Terminology).

Projects with a known Defects4J problem (Cli): In the current Defects4J 3 release, every EvoSuite test for the Cli project fails, on both the buggy and the fixed versions, with java.lang.NoClassDefFoundError: org/hamcrest/SelfDescribing, because framework/projects/Cli/Cli.build.xml refers to a JUnit jar that does not exist. If you see this, edit that file in your Defects4J installation and replace framework/projects/lib/junit-4.12.jar with framework/projects/lib/junit-4.12-hamcrest-1.3.jar. If this problem affected your Homework 2 results (for example, your suites' tests failed on the fixed version, or fix_test_suite.pl printed warnings such as "Use of uninitialized value" instead of removing tests), re-run what was affected, and describe it in your README.

Fixing suites: If you need to remove failing or flaky tests from a suite again, fix_test_suite.pl is in framework/util (not framework/bin).

Statistics: You may use any statistics package. In Python, SciPy provides what you need:

- Point-biserial correlation: scipy.stats.pointbiserialr(detection, score).
- Rank-biserial correlation: compute the Mann-Whitney U statistic for the $\tilde{T}_{fail}$ group with scipy.stats.mannwhitneyu(fail_scores, pass_scores). Then $r = 2U/(n_{fail} \cdot n_{pass}) - 1$, and the Mann-Whitney test's p value is the p value for r.
- Wilcoxon signed-rank test: scipy.stats.wilcoxon(x, y, alternative='greater') tests whether the values in x tend to be greater than the paired values in y. If all paired differences are zero, the test cannot be computed; report that.
- If every suite of a bug has the same mutation score (or statement coverage), its correlation coefficient is undefined; report that.

Output files get overwritten: As in Homework 2, Defects4J 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 loops over your suites and copies the results into <Project>-<BugID>/detection/suiteNN/ and <Project>-<BugID>/coverage/suiteNN/ will save you time and mistakes.