Test Adequacy via Mutation Analysis

For this project, you shall perform a mutation analysis study of an existing test suite for an open-source software project. We shall be using the Major mutation framework for Java to perform the actual mutation analysis.

You shall work in groups of three to four students, but the overall requirements scale with the group size.

Selecting a Project to Analyze

The following requirements apply to the open-source project.

1. It must be hosted/sponsored/associated with an organization that is affiliated with Google Summer of Code (in this or any prior year), or it must be present in the Open Hub database. Exceptions to this are possible but must be approved by the instructor. (You might look at the programs Major was tested on for ideas.)
2. The code base of the analyzed project must be at least 10,000 lines of code (10 KLoC). The sizes of most projects can easily be found via OpenHub or via tools like `sloccount`.
3. A substantial test suite must exist for the project. The assignment will be easiest if the selected project uses JUnit to manage its tests.

Assignment Project Exam Help

In finding which project to analyze, you should identify and consider the following attributes:

https://powcoder.com

1. Identification of the open-source project.
2. Identification of the supporting organization.
3. Size of the code base (lines of code).
4. Proposed evaluation platform (OS, language).
5. Build time to compile and link an executable from source code.

Add WeChat powcoder

6. Test suite infrastructure (JUnit?, Ant integration?).
7. Number of tests, lines of code, & execution time for the test suite.

Make sure to include this information in your reported results. If you have questions about whether a particular project is a good choice, identify these attributes and *ask*.

A key part of this task is making sure that you can consistently compile the project and run its test suite.

Running the Major Mutation Framework

Major provides custom versions of Java tools like `javac` and `ant` that transform a program to inject mutations. Many mutations are injected into the program at once, but they can be enabled individually when tests run. For Major to perform mutation analysis on a project, that project must first be compiled using the provided version of `javac`, and then the test suite must be executed using additional arguments that enable Major's analysis and reporting. This means that you will need to integrate Major into the build tools used by the project you select. Presently, should be used from the command line.

Every project is a bit different, and integrating Major with your project's test suite may involve adapting, modifying, or otherwise changing the build system in order to use Major. For projects that use `javac` and `java` directly or for projects that use `ant` to manage building and testing a project, examples are available here and are described in section 5 of the documentation.

Projects using Maven or other build systems are easiest to use by first exporting an `ant` build.xml file and modifying it (via `mvn ant:ant` for Maven).

To integrate Major with an `ant` build.xml file, there are a few simple steps to follow:

1. Add the property definitions used by Major at the top of the `ant` project. You will need to replace `<path to major>` with the actual path where Major is located.

```
<property name="mutation" value=":ALL"/>
<property name="mutator" value="-XMutator${mutation}"/>
<property name="major" value="<path to major/major/bin/javac>"/>
```

2. Find the `javac` task/tag within the "compile" target and add the attribute `executable="${major}"`. Afterward, add the element `<compilerarg value="${mutator}"/>` within the matching `<javac>` and `</javac>` tags.

3. Similarly, find the `javac` task within the task that compiles the tests, e.g. `compile-tests`, and add the `executable="${major}"` attribute. Do *not* add the mutator element to this.

4. Find the `junit` task within the "test" target and add these attributes:

```
mutationAnalysis="true"
resultFile="results.csv"
killDetailsFile="killed.csv"
```

Assignment Project Exam Help

An example build.xml with these modifications is provided with Major and is also available here. After modifying the `ant` build file, compile and run the tests using the modified version of `ant` at `<path to major>/major/bin/ant`. Again, expect that you may have to slightly modify these instructions to have them work for a particular project.

https://powcoder.com

Add WeChat powcoder

After running the mutation analysis with these modifications to the `ant` build script, several new files are created. `mutants.log` contains one line for each generated mutant. Each line contains the type of mutation, the method and line of the mutation, and the actual syntactic mutation applied. `killed.csv` also contains a line for each generated mutant. Each line identifies the mutant ID from `mutants.log` and the status of the mutant at the end of analysis. `FAIL`, `TIME`, and `EXC` mean that the mutant was killed by an assertion failure, test timeout, or an exception. `LIVE` means that the mutant was still live at the end of analysis. `results.csv` contains the mutation analysis for each individual test in the test suite. Finally, `summary.csv` contains the complete mutation analysis results. Other useful information can also be exported to files as described in the Major documentation.

Note: Just *running* mutation analysis can take a long time, even hours for some projects. Recall from our class discussion why this is. Plan ahead to make sure you have enough time.

Group Analysis and Results

The group as a whole will perform mutation analysis for the entire project using the Major mutation framework. In reporting your results, you should include:

- The number of mutants generated
- The number of mutants *covered* by the test suite
- The number of mutants *killed* by the test suite
- The number of *live* mutants

- The overall mutation score / adequacy of the test suite

Discuss and explain your methods and your observed results. Does `killed + live = covered` or `... = generated`? Why or why not? What do the results tell you about your test suite? Does the test suite exhibit weaknesses? How can it be improved? Does the test suite exhibit strengths? How do you recognize them? Contrast the costs and benefits of mutation analysis and testing versus what you might expect from other techniques. What was easy? What was difficult? What obstacles did you face in applying mutation analysis to a real world project, and how did you overcome them? Do you have any other interesting insights or opinions on the experience?

Note, the above points are not yes or no questions. You should present evidence and justify your answers.

Individual Analysis and Results

Each group member shall also choose one method from the open source project and consider the results of the mutation analysis specifically for that method. Recall that these mutants are available to you in the `mutants.log` file generated by `Major`. Choose a complex routine that you think is likely to have errors. If `Major` generated fewer than 25 mutants for the method, then either select a *different* method or select *additional* methods until you have 25 mutants to consider.

Examine 25 of the generated mutants for your method(s). If both killed and unkilled mutants were generated, include a mix of both. For each one, document your mutation operator. What was the type of operator used? How was it applied to the code (how did the code change)? Include the actual line from `mutants.log` in addition to your descriptive documentation.

Assignment Project Exam Help
https://powcoder.com
Add WeChat powcoder

For your individual mutants consider the following additional questions: How many mutants are killed? How many mutants are live? Does `live+killed=25`? Why or why not? For every mutant that was not killed, try to determine either (a) that it is an equivalent mutant that should not be killed, or (b) how to add a test to kill it. For all mutants, try to determine whether they are duplicates of each other. What are the challenges involved? Does it affect the results? Calculate and report the mutation score / effectiveness for your particular mutants. What do these results say about the effectiveness of the test suite and the method(s) that you condidered?

Submission

Each group shall submit a final writeup including descriptions of the selected project, the group results, and the individual results as discussed above. Also include the `mutants.log`, `killed.csv`, `results.csv`, and `summary.csv` files produced by the Major framework.

The assignment is due 11:59 pm on Wednesday, March 7.