PIT Mutation Tool Additions

Matthew Bachelder
University of Texas at Dallas
United States
mtb140130@utdallas.edu

Vaishnavi Bhosale
University of Texas at Dallas
United States
vbb160030@utdallas.edu

Richard Fisher
University of Texas at Dallas
United States
rxf120030@utdallas.edu

ABSTRACT

Mutation testing is a method of software quality control that tests the unit test suite defined for programs by artificially injecting errors, or mutations, into software. Tests are then executed to evaluate the quality of the test suite. If the mutations are detected, then the mutation is killed and the test suite is deemed adequate with respect to that section of code. If the mutation is not detected, then it is said to have passed, and the test suite for that section of code is inadequate.

CCS CONCEPTS

• **Software engineering → Software testing and debugging**;

KEYWORDS

PIT Mutation testing, Mutators, Operators

1 INTRODUCTION

Mutation testing is a method of software quality control that tests the unit test suite defined for programs by artificially injecting errors, or mutations, into software. Tests are then executed to evaluate the quality of the test suite. If the mutations are detected, then the mutation is killed and the test suite is deemed adequate with respect to that section of code. If the mutation is not detected, then it is said to have passed, and the test suite for that section of code is inadequate.

Mutations, or faults that are injected into code, are modifications to various operators within the code. There are numerous mutations that can be used but commonly used examples include:

- Modification of increment operators to decrement
- Negation of certain values
- Modification of constants
- Modification of comparison operators

2 Proposed Additions to PIT

This project will add several mutations to the suite of available code modifications. These include:

a. ABS: Replaces a variable by its negation, e.g., a becomes –a
b. OBBN: Replaces the operators & by | and vice versa, e.g., a&b becomes a|b
c. AOD: Replaces an arithmetic expression by one of the operand, e.g., a + b becomes a
d. ROR: Replaces the relational operators with another one. It applies every replacement, e.g., < becomes ≥, or > becomes ≤
e. AOR: Replaces an arithmetic expression by another one. a + b becomes a ∗ b
f. UOI: Replaces a variable with a unary operator or removes an instance of an unary operator. a becomes a++
g. CRCR: Replaces a constant a with its negation, or with 1, 0, a + 1, a – 1, e.g., a becomes −a, and a becomes a − 1.

3 PIT: REAL WORLD MUTATION

PIT is a mutation testing framework for Java developed to support the day to day development on real codebases. This means that PIT aims at: (1) having a good integration with build tools like Maven, Ant, Gradle and integrated development environments like Eclipse or IntelliJ. (2) being fast: PIT uses three techniques to obtain its results:

working on bytecode instead of source code, selecting the tests to run against the mutants and minimizing the number of mutant executions. (3) making a clear report of the tests execution. This makes the navigation between source code and mutants easy by highlighting mutants that were not killed.

4 Experimental Evaluation

The proposed modifications to PIT will be evaluated on five real world projects chosen from GitHub. Each project will contain a minimum of 1,000 lines of code, and will be evaluated after running at least 50 tests per project. The overall quality of the test suites, when tested with the augmented PIT, will be measured using the mutation adequacy score given in equation (1).

$$MAS\ (P, TS) = \frac{\mathrm{Kn}}{M - En} \qquad (1)$$

Where, P = Program under test, TS = Test suite, Kn = number of Killed mutants, M = total number of Mutants, En = number of equivalent mutants.

5 Progress

This section is meant to update on how the project was progressing at the time of our midterm progress report submission. The format is meant to highlight evaluation and procedural steps we that have been completed. A separate section, section 6, will discuss our plans for completing the remainder of the project.

REPOSITORY

For purposes of collaboration, and eventual project evaluation, we were required to choose a repository that would store all of our project assets. Assets included in the repository are source code for the augmented PIT utility, and any documentation created for the purpose of completing this project.

The repository we chose to use was git, rather than Subversion or TFS because the entire team was familiar with git, and because PIT is housed at Github. This allowed us to use a single repository for all our work. In addition, everyone in the group was familiar with it and we would not have to spend a lot of effort on learning a new repository.

The project guidelines also required we host our repository, so that everyone in the group could access the most up to date source code and documentation, and also so the project could be evaluated by the TA on submission.

Everyone in the group has accessed the repository and built a local copy on their machines. The repository can be accessed by its members at https://github.com/matthewbachelder/cs6367.git

PLANNING

After establishing a repository and ensuring all members in the group were able to both access the repository and build a local copy on their machines, we decided to evaluate the best order of steps to complete the remainder of the project as efficiently as possible. We have added planning as a subsection to Progress to underscore the importance of it. This subsection will only include the steps planned and executed up until the time the midterm progress report was submitted. Steps that are planned after the submission of the progress report will be added in section 6 of this report.

1. Explore pitest.org and become familiar with the PIT tool documentation and any resources available.
2. Download PIT and all dependencies
3. Ensure we are able to build a modified version of PIT

EXPLORE PITEST.ORG AND BECOME FAMILIAR WITH THE PIT TOOL DOCUMENTATION AND ANY RESOURCES AVAILABLE

The purpose of exploring pitest.org and the documentation for the PIT utility, was to get familiar with what resources would be available to us for the duration of the project and where those resources were located. Knowing what is available and where it is located will be crucial to the efficient implementation of this project. A brief overview of the section we noted as potentially useful follows

1. Quickstart
   a. The quick start section of pitest.org contains all of the information we will need to build and run the PIT utility on our local machines. It includes guides for Maven, Command Line, Ant, and Gradle. Each quick start guide links to specific instructions for using the listed frameworks.
2. Basic Concepts
   a. An overview of the different mutation operators supplied by PIT and a description of what each one can achieve. This section also includes information on what to expect when evaluating the results of tests using PIT and how to handle common errors, memory and runtime, that may occur. Finally this section concludes with a description of an experimental feature called

'Incremental Analysis' which may be useful for our project depending on the test project size.

3. Available Mutators
   a. A more detailed section at pitest.org with every mutator implemented by PIT with a detailed description of each one and what is achieved by their use.

DOWNLOAD PIT AND ALL DEPENDENCIES

The next important step, after taking some time to learn about the resources available to us for PIT, was to download a copy of PIT and any dependencies that would be required to build it.

To complete this step we first needed to locate a copy of the PIT source code. Pitest.org contains a link to the git repository where PIT is hosted. Using this we were able to clone a copy of the source code from a repository owned by hcoles located here https://github.com/hcoles/pitest.git.

After downloading the copy of PIT we removed the git files from the repository and moved the source code over to a group member's local repository before pushing it to our groups remote repository, so that other members could download from there. Doing this also ensured that everyone in the group was using the same copy of PIT, potentially other modified versions exist, and any modifications to PIT made by anyone in the group could be made available to everyone in the group through the group repository.

Dependencies for PIT were installed by opening the PIT project in the IntelliJ IDE and pointing to the pom.xml file, so that they could be downloaded using Maven. Doing it this way ensured that all dependencies would be accounted for and we would not have to locate and install each one manually.

ENSURE WE ARE ABLE TO BUILD A MODIFIED VERSION OF PIT

After downloading PIT we needed to be sure we were comfortable with the build process that would need to be completed after making the modifications necessary to augment with additional operators.

To complete this a group member located the existing mutation operators in the PIT source code and completed two of the additions from our original proposal.

After modifying the source code for PIT and adding two additional operators the group member was able to rebuild PIT using tools available in the IntelliJ IDE. The build was successful, so at this point we assume we have what we need to continue our work on this project and list below, in section 6, the next steps to do so.

5 Further Plan

The work we have completed to this point in the project has been done to assist in collaboration and to become comfortable with PIT and the documentation and resources available to use for PIT. Our next steps are aimed at completing the augmentation of PIT and finally running more than 50 tests with it on no less than five real world projects from Github with at least 1000 lines of code or more. Steps we will take to complete the remainder of this project follow.

1. Choose a project off of Github to test the modified PIT build on.
2. Complete the augmentation of PIT
3. Identify 7-10 real world projects that will be used to evaluate the augmented PIT.
4. Complete and log evaluations on 5 of the real world projects chosen in C.
5. Report on our findings.

CHOOSE A PROJECT OFF OF GITHUB TO TEST THE MODIFIED PIT BUILD ON

The purpose of this step is to find a small project that we can use to validate the modifications we make to the PIT source code. This will help us to eliminate the need to debug our additions when testing on the real world projects that will be included in our results.

COMPLETE THE AUGMENTATION OF PIT

To complete this step we intend on adding all of the additions included in our original proposal to PIT.

IDENTIFY 7-10 REAL WORLD PROJECTS THAT WILL BE USED TO EVALUATE THE AUGMENTED PIT

We will use this step to locate viable projects from Github that may be used in our final results. We intend on locating 7-10, while only 5 are required, because we want to be sure we have additional projects ready if one of the five we need to use is not compatible with PIT.

COMPLETE AND LOG EVALUATIONS ON 5 OF THE REAL WORLD PROJECTS CHOSEN IN C

This step is to complete the evaluations of the real world projects using our augmented PIT. Evaluations will be logged as noted in our original proposal.

REPORT ON OUR FINDINGS

In this step we compile our logs into a final report intended for submission.

REFERENCES

[1] [11] H. Coles. 2017. “PIT.” Retrieved from http://pitest.org/.

[2] Coles, Henry, Thomas Laurent, Christopher Henard, Mike Papadakis, and Anthony Ventresque. "PIT: a practical mutation testing tool for Java." In Proceedings of the 25th International Symposium on Software Testing and Analysis, pp. 449-452. ACM, 2016.

[3] Rani, Shweta, et al. “Experimental Comparison of Automated Mutation Testing Tools for Java.” *2015 4th International Conference on Reliability, Infocom Technologies and Optimization (ICRITO) (Trends and Future Directions)*, 2015, doi:10.1109/icrito.2015.7359265.