# Analysis of Mutation Testing Tools

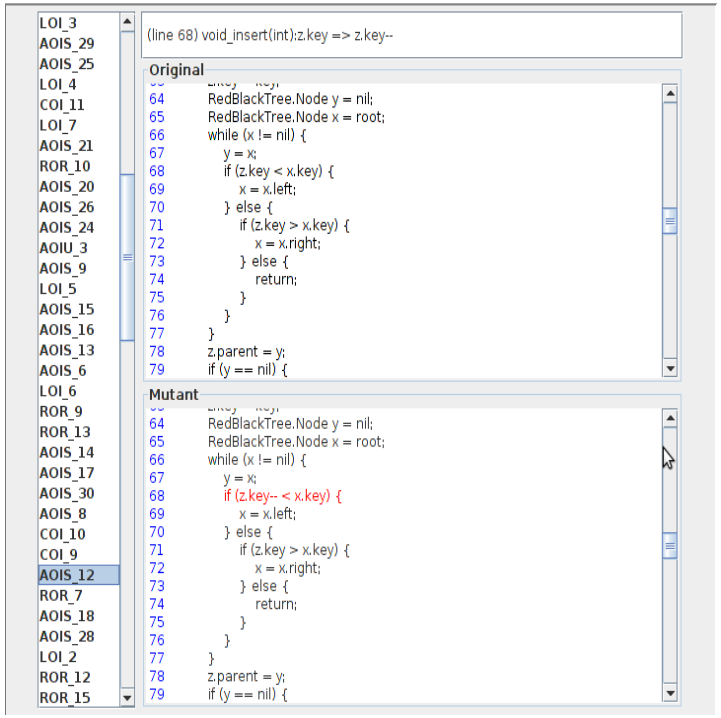
Johnathan Snyder, *Department of Computer Science, University of Alabama*
Adviser: Jeff Gray, *Department of Computer Science, University of Alabama*
Contact: jrsnyder@crimson.ua.edu

THE UNIVERSITY OF ALABAMA COMPUTER SCIENCE

## Introduction

Software testing is an important component of software development because it helps determine if there are any faults in the software and if the software meets its requirements. Research in software testing seeks to improve different testing techniques. Mutation testing is a software testing technique that tests the quality of test suites by generating mutants of the source code and running the mutants against the test suites to see if they have different results. However, mutation testing is not widely used in industry because it is computationally expensive and because some mutants are syntactically different from the original code but semantically the same. The development and improvement of mutation testing tools will help solve these problems and will help lead to more industry adoption of mutation testing. The improvements of these tools can be built upon the technologies of current mutation testing tools. This research reports on a study that was made on mutation testing tools for Java programs. Each of the mutation testing tools was analyzed to see how efficiently they could generate mutants, what type of mutants were generated, and how well they could integrate with other testing tools such as JUnit.

## Example Mutant Generated by MuJava




## Mutation Testing Process

The Mutation testing process starts after the programmer has written the code to be tested, as well as the unit tests for that code. The first thing is to execute the original program against the unit tests. Then, mutant programs are generated from the original code and the equivalent mutants are weeded out using a heuristic algorithm. As a next step, each mutant is executed against the test cases to see if the test cases can catch the error. If the test cases catch an error, then the mutant is said to be killed; otherwise, the mutant is alive. After all the mutants have been run against the test cases, the mutation score is calculated by the equation "score = (100 * D) / (N – E)" where D is the number of dead mutants, N is the total number of mutants generate, and E is the number of equivalent mutants. The score is between 1.0 and 0.0, with a 1.0 indicating the best score.

## Efficiency

A major problem for mutation testing tools is deciding how to generate mutants and run them against the test suite in an efficient way. One way to approach this is to compile all the mutants generated and run them against the test suites. However, compiling a program and all its mutants is very computationally expensive. Another way is to implement an interpreter in the mutation testing tool, but interpreters are also very slow and just as much computationally expensive as a compiling every single mutant. The most efficient way is bytecode mutation. Languages like Java do not compile directly to machine code, but are compiled to a platform-independent bytecode where the bytecode is executed on a virtual machine. This both reduces the time to compile the program, since it only has to be compiled once, and run the mutants against the test suites since virtual machines are more efficient than interpreters. All three of the mutation tools that I studied (MuJava,Jumble,PIT) used bytecode mutation.

## Mutation Operators

| Operator | Description |
|---|---|
| AOR | Arithmetic Operator Replacement |
| AOI | Arithmetic Operation Insertion |
| AOD | Arithmetic Operation Deletion |
| ROR | Relation Operator Replacement |
| COR | Conditional Operator Replacement |
| COI | Conditional Operator Insertion |
| COD | Conditional Operator Deletion |
| SOR | Shift Operator Replacement |
| LOR | Logical Operator Replacement |
| LOI | Logical Operator Insertion |
| LOD | Logical Operation Deletion |
| ASR | Assignment Operator Replacement |

## Conclusion

In my research, I studied three mutation testing tools for Java: MuJava, Jumble, and PIT. All of them use byte level mutation which speeds up the time it takes to generate the mutants and run the mutants against the test suite. Both Jumble and PIT have support for JUnit, the Java unit testing framework. However, PIT is the most capable for use in industry because it also integrates well with other Java tools such as ant and is the most actively supported project.

## Bibliography

Amman, P. & Offutt, J. (2008), *Introduction to Software Testing*. NewYork: Cambridge University Press

Kwon Y. Ma, Y. & Offut, J. (2005). MuJava: An Automated Class Mutation System. *Journal of Software Testing, Verification and Reliability,*15(2), 97-133