

Software Testing Report

Mani Nandadeep

IMT2019051

G Sri Harsha

IMT2019030

November 28, 2022

Abstract

The following document is a report of the details of the concepts involved in the project of CS731 Software Testing Course.

Keywords: Mutation Testing, PITest, JUnit, Unit Testing

1 Problem Description

Problem Statement: Mutation source code: Projects that use mutation testing, based on mutation operators applied at the level of a statement within a method or a function.

Aim: The mutated program needs to be strongly killed by the designed test cases. At least three different mutation operators should be used.

2 Introduction

Mutation Testing is a type of Software Testing that is performed to design new software tests and also evaluate the quality of already existing software tests. Mutation testing is related to modification a program in small ways. It focuses to help the tester develop effective tests or locate weaknesses in the test data used for the program.

Mutation testing involves making syntactically valid changes to a software artifact and then testing the artifact. We consider grammars of software artifacts again. Grammars generate valid strings. We use derivations in grammars to generate invalid strings too. Testing based on these valid and invalid strings is called mutation testing

3 PITest

PIT is a state of the art mutation testing system, providing standard test coverage for Java and the jvm. It's fast, scalable and integrates with modern test and build tooling.

PIT runs unit tests against automatically modified versions of the application code. When the application code changes, it should produce different results and cause the unit tests to fail. If a unit test does not fail in this situation, it may indicate an issue with the test suite.

4 Source Code

Link to source code - github.com/GSri30/Software-Testing-Project

Link to PITest results - gsri30.github.io/Software-Testing-Project/

A linked list is a linear data structure, in which the elements are not stored at contiguous memory locations. The elements in a linked list are linked using pointers.

Some of the main functionalities implemented are:

1. `add(int)` : This function adds a data node to the linked list at the end of the list thereby, increasing the size of the list by 1.
2. `add(int, int)` : This function adds a data node to the linked list at a given valid index.
3. `addFirst(int)` : This function adds a data node at the head of the linked list.
4. `addLast(int)` : This function adds a data node at the tail end of the linked list.
5. `remove(int)` : This function removes the data node from the provided index of the linked list.
6. `removeNodeGivenAsReference(Node)` : This function removes the data node given by the reference. It uses the address pointer to remove the node.
7. `removeDuplicatesInSortedList()` : This function removes the duplicate data value nodes from the sorted linked list.
8. `removeDuplicatesInUnsortedList()` : This function removes the duplicate data value nodes from the unsorted linked list.
9. `removeAlternateNodes()` : This method removes the alternate data nodes from the linked list.
10. `reverseByK(int)` : This method reverses linked list elements in the group of given number k.
Example: 1->2->3->4->5->6->7 Input: K=3 Output: 3->2->1->6->5->4->7
11. `reverseAlternateKNodes(int)` : This function reverses the alternate k linked list nodes.
12. `get(int)` : This method returns the desired data (if present) from the list.
13. `getMiddleElement(Node)` : This method returns the data node value which is present in the middle of the linked list.
14. `printMiddleElement()` : This method prints the data node value which is present in the middle of the linked list.

15. `printNthNodeFromLast(int)` : This method prints the Nth node present from the tail end of the linked list.
16. `isPalindrome()` : This method returns boolean whether the linked list is a palindrome or not.
17. `mergeTwoSorted(ArrayLinkedList, ArrayLinkedList)` : This method merges two sorted linked lists and returns the new linked list.
18. `isIdentical(ArrayLinkedList, ArrayLinkedList)` : Returns boolean whether the 2 lists are completely identical or not.
19. `swapPairWise()` : This function swaps the data nodes pair wise.
Input: 1->2->3->4 Output: 2->1->4->3

5 Mutation Operators

Some of the active mutators present were:

- **Negate Conditionals Mutator** The negate conditionals mutator will mutate all conditionals found according to the replacement table below.

Original conditional	Mutated conditional
<code>==</code>	<code>!=</code>
<code>!=</code>	<code>==</code>
<code><=</code>	<code>></code>
<code>>=</code>	<code><</code>
<code><</code>	<code>>=</code>
<code>></code>	<code><=</code>

Figure 1: Negate Conditional Mutators

- **Math Mutator**
The math mutator replaces binary arithmetic operations for either integer or floating-point arithmetic with another operation. The replacements will be selected according to the table below [2](#).
- **Increments Mutator**
The increments mutator will mutate increments, decrements and assignment increments and decrements of local variables (stack variables). It will replace increments with decrements and vice versa.
For Example: `i++` will be mutated to `i--`;
- **Void Method Call Mutator** The void method call mutator removes method calls to void methods. For example refer to figure [3](#) and figure [4](#).

Original conditional	Mutated conditional
+	-
-	+
*	/
/	*
%	*
&	
	&
^	&
<<	>>
>>	<<
>>>	<<

Figure 2: Math Mutators

```

public void someVoidMethod(int i) {
    // does something
}

public int foo() {
    int i = 5;
    someVoidMethod(i);
    return i;
}

```

Figure 3: before void method call mutation

```

public void someVoidMethod(int i) {
    // does something
}

public int foo() {
    int i = 5;
    return i;
}

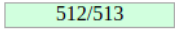
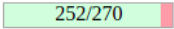
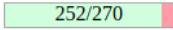
```

Figure 4: after void method call mutation

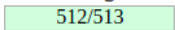
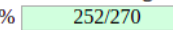
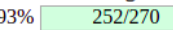
6 PITest Results

Pit Test Coverage Report

Project Summary

Number of Classes	Line Coverage	Mutation Coverage	Test Strength
1	100% 	93% 	93% 

Breakdown by Package

Name	Number of Classes	Line Coverage	Mutation Coverage	Test Strength
com.mylinkedList.app	1	100% 	93% 	93% 

Report generated by [PIT](#) 1.9.11

Enhanced functionality available at [arcmutate.com](#)

Figure 5: PITest Results

7 Execution

7.1 Pre-Setup Required

- Make sure to have Java JDK 11 successfully setup.
- Also make sure to have maven installed with proper PATH variables initialized.

7.2 How to Run

Run the following command:

```
mvn install
mvn test-compile org.pitest:pitest-maven:mutationCoverage
```

7.3 Contributions

Mani Nandadeep:

- Worked on the other section of source code and mutation tests.
- Worked on curating the final project report.

G Sri Harsha:

- Worked on setting up maven framework environment, JUnit4 and PITest initialization.
- Worked on writing some sections of source code and mutation tests.

References

1. *Maven and JDK setup* at <https://www.digitalocean.com/community/tutorials/install-maven-linux-ubuntu>

2. *Unit testing with JUnit4* at <https://www.vogella.com/tutorials/JUnit4/article.html>
3. *PITest mutation coverage documentation* at <https://pitest.org/>