UTM Computing Proceedings
Innovation in Computing Technology and Applications
Volume: 2| Year: 2017 |ISBN: 978-967-0194-95-0

A Comparative Study on Mutation Analysis for Java Applications

Nur Zatilzaida Binti Mohd Zahari[1] and Radziah Binti Mohamad[2]

Department of Software Engineering, Universiti Teknologi Malaysia, Skudai, Johor, Malaysia
[1]zatilzaida93@yahoo.com, [2]radziahm@utm.my

Abstract. *Java applications have the highest numbers of users. However, the quality of the java applications is a known problem. Thus, the better testing has been research to apply on java applications for better quality which is mutation analysis for java applications. First, identify the state-of-the-art mutation technique for java applications. Second, identify the mutation operators for java applications that will make the java applications have good quality. Finally, the suitable mutation operators will be implementing to the java applications test case. This is particularly problematic because lack of approach of mutation analysis for java applications. For these reasons, this research proposes suitable mutation operators for java applications.*

Keywords : *Mutation Testing, Java Applications, Software Testing*.

7. 1 Introduction

When programmers develop a java applications there is some testing need to be done. The testing is to check whether the applications fulfill all the requirements and suitable to the user. The problem is some testing is ineffective. There is some faults can affect the applications in the future that cannot be capture during testing phase. The solution is to use mutation testing on java applications which is still immature and rarely used.

Testing java applications using an automated testing tool between various mobile platforms has been conducted, but the results do not always match the actual mobile network, a network that makes the test becomes weaker and normal test does not support more than one smart device for testing java applications for mobile [10]. Testing mobile applications have many limitations according to the following issues which is different mobile operating system, mobile network traffic and variety of type of mobile devices. Furthermore, testing conduct on context-based on java applications is not many have been done. In addition, testing on context-based on java applications is also difficult to found.

A variety of mobile platforms, limitations in the simulator, the boundaries of mobile hardware devices, multiple network types and configurations, and rapid application development of mobile applications make the java applications test become more challenging, time consuming and expensive process [1].

In particular, this research will propose to use the mutation analysis to java applications, high-end testing techniques known to produce test powerful and robust [5]. The advantages and limitation will be compared and documented. The comparison between the different approaches of mutation testing but need to find the same fault will be documented. Mutation testing always use for testing on web applications not for both web and mobile applications which is java applications used to developed the applications. Thus, java applications are different from traditional software and others applications that use others language. When apply mutation testing on java applications, need to develop different testing technique because of the mutation testing on java applications still immature and lack of capability to support the testing.

2 Related Works

Mutation testing will be applied to the java applications where the mutation testing is high end testing technique that is known for yielding powerful tests. Test than kill those mutants expected to have many faults in the use of the features [5].

In this research, mutation testing will be reviewed from other related works and the suitable mutation testing technique and operators will be selected. The selected mutation testing technique and operators will be implements to the case study which is java applications. The result will determine the quality of the testing and the quality of the java applications.

The existing works is the review about others work on mutation testing analysis. The review is about the mutation testing for applications that has been developed. The review of the applications is divided to web applications and mobile applications using java language and it is a java applications.

a. Web Applications

According to others [14], to get the strong test criteria, mutation analysis is fault-based techniques that provide the strong test criteria. This technique injects artificial fault into the software that being tested. Fault injection made by the mutation operators that represent the type of fault that want to explore. To expose the damage in the JS program, we need to define the mutation operator with a focus on the characteristics of the JavaScript and in order to conduct an analysis of the characteristics of the JavaScript; we define the mutation operator based on the analysis results [14].

To identify the common mistake in applications we suggest the following JavaScript -specific mutation operator to understand the common mistakes in JavaScript programs from the point of view of the programmer [12].

In conclusion, in order to implement mutation testing to the web applications, some mutation operators need to be defining based on the characteristics of the JavaScript.

b. Mobile Applications

Although mutation testing is one way to create more effective test suite, along with error detection to a new level for software developers, which creates a more effective test suite, testers can be more confident that the program sufficiently tested [17]. The categories of mutation operators for Java Agent Development framework (JADE) Mobile Agent System (MAS) are: Agent setup and creation, Mobility, Agent message passing, and Agent behavior [17]. Table 1 Show the list of JADE mutation operators [17].

Table 1 : JADE MUTATION OPERATORS

Operator category	Mutation Operator Description
Agent setup and creation	RSM: Remove setup method.
	RSMWO: Replace setup method with any other method.
	MCNAP: Modify createNewAgent object pararmeter.
	MCNFP: Modify createNewAgent file name pararmeter.
Mobility	CDMD: Change doMove Destinations.
	IWBM: Insert doWait statement before doMove.
	IDSBM: Insert doDelete statement before doMove.
	RANMS: Remove agent name from doMove statement.
	SDMS: Switch doMove statement.
	RDMS: Remove doMove statement.
	RDMOS: Replace doMove with other statement.
	DMOSS: Put doMove statement out of switch statement.
	RAMS: Remove afterMove statement.
Agent Message passing	RGM: Remove get method.
	CMC: Change message content.
	RSS: Remove send statement.
	RARS: Remove addReceiver statement.
	RRP: Remove receives parameter statement.
	RRRS: Replace receive with blockingReceive statement.
	CBRP: Change blockingReceive parameter.
	MCMIE: Move Communication method with IfElse.
	RNUM: Remove notifyUser method.
Agent behavior	RABM: Remove addBehaviour method.
	ROEM: Remove onEnd method.
	ROTM: Remove onTick method.
	RHETOM: Remove handleElapsedTimeout method.

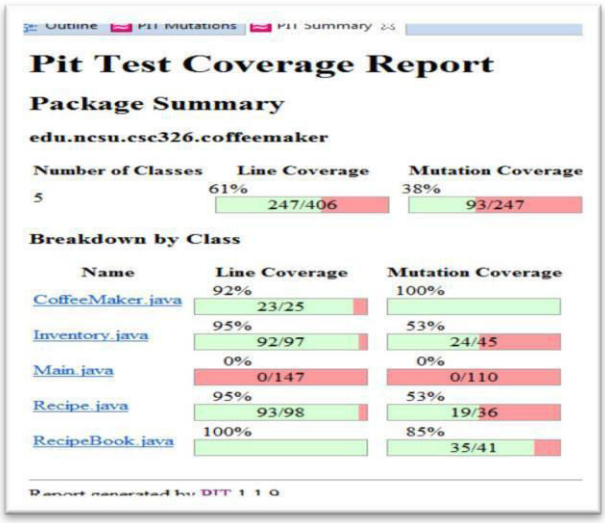
From this finding, some of the mutation operators and technique from the existing works can be derive and implement to the case study. The suitable mutation operators will be selected from the related works and will be implemented to the case study.

8. 3 Result

Mutation testing experiment for this research is based on the mutation testing for java applications. The java applications case study that has been used as an experiment is Coffee Maker applications. The mutation testing tools used to carry out the experiment for mutation testing for java applications is Pitest mutation testing tools. It is a good test if the mutation coverage for the test cases is 100%.

a Pitest Coverage Report

Further discussion of the result for the mutation testing experiment on java applications will be discuss and analyse. Fig 1 shows the Pitest coverage report for mutation testing on Coffee Maker applications. The Pitest coverage report shows the percentage of the line coverage and mutation coverage. In this report it shows that line coverage is only 61%, which means some of the class in the applications where the line of code is not cover during mutation testing. Meanwhile for mutation coverage it only covers 38%. It means that our test is not as good as it should be. In order to achieve the good mutation testing, some changes in the mutation test cases need to be done until the mutation coverage reach 100% coverage.



Pit Test Coverage Report

Package Summary

edu.ncsu.csc326.coffeemaker

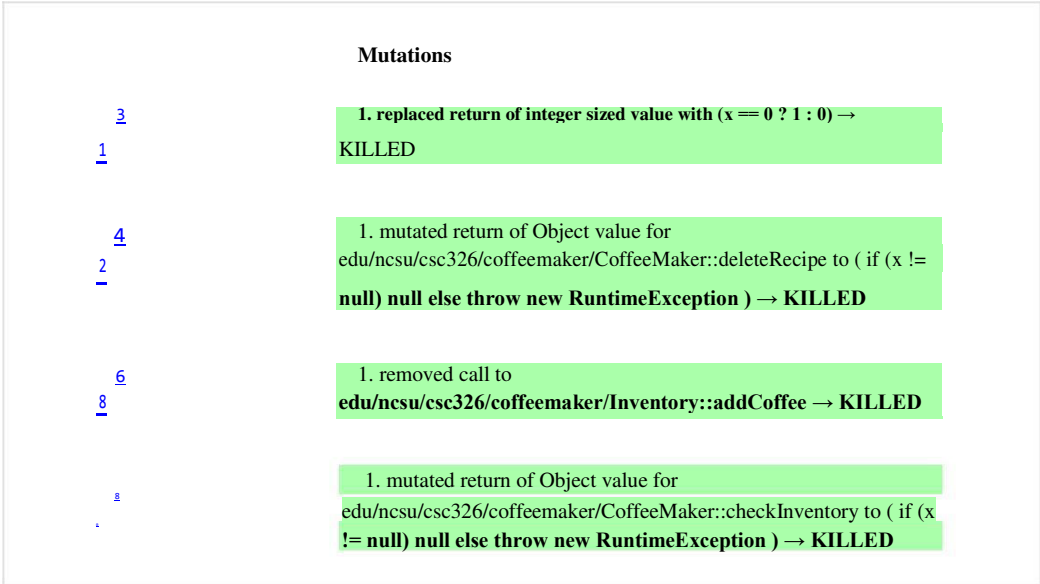
Number of Classes	Line Coverage	Mutation Coverage
5	61% 247/406	38% 93/247

Breakdown by Class

Name	Line Coverage	Mutation Coverage
CoffeeMaker.java	92% 23/25	100%
Inventory.java	95% 92/97	53% 24/45
Main.java	0% 0/147	0% 0/110
Recipe.java	95% 93/98	53% 19/36
RecipeBook.java	100%	85% 35/41

Figure 1. Pit Test Coverage Report for Coffee Maker.

i. Coffee Maker Class Report: Fig 2 shows CoffeeMaker.java mutation report where the mutation coverage is 100%. The mutation coverage is 100% because the mutation test case for CoffeeMaker.java is all killed and it means that our mutation test cases for CoffeeMaker.java are good test cases.



Mutations

3 1 **1. replaced return of integer sized value with (x == 0 ? 1 : 0) →** KILLED

4 2 1. mutated return of Object value for edu/ncsu/csc326/coffeemaker/CoffeeMaker::deleteRecipe to (if (x != **null) null else throw new RuntimeException) → KILLED**

6 8 1. removed call to **edu/ncsu/csc326/coffeemaker/Inventory::addCoffee → KILLED**

1. mutated return of Object value for edu/ncsu/csc326/coffeemaker/CoffeeMaker::checkInventory to (if (x **!= null) null else throw new RuntimeException) → KILLED**

Figure 2. CoffeeMaker.java mutation report.

ii. *Main Class Report:* Fig 3 shows Main.java mutation report where the mutation coverage for mutation testing is 0%. This is a very bad mutation test cases for mutation testing because the

mutation test cases is not all killed and it is all not covers by the mutation testing. It has no mutation coverage because of the error when derived the mutation test cases. In order to avoid this happen, many changes need to be done to the mutation test cases until the problem of no mutation coverage is solved. Then, mutation testing can be carry again until the mutation coverage is 100% for the best mutation testing. It is also important to make sure the mutation test case is suitable to the case study applications for mutation testing. When derived the mutation test cases, need to make the mutation test cases is not have any errors in coding so that the line of code of the mutation code will cover the mutation line coverage and this problem has never occurred.

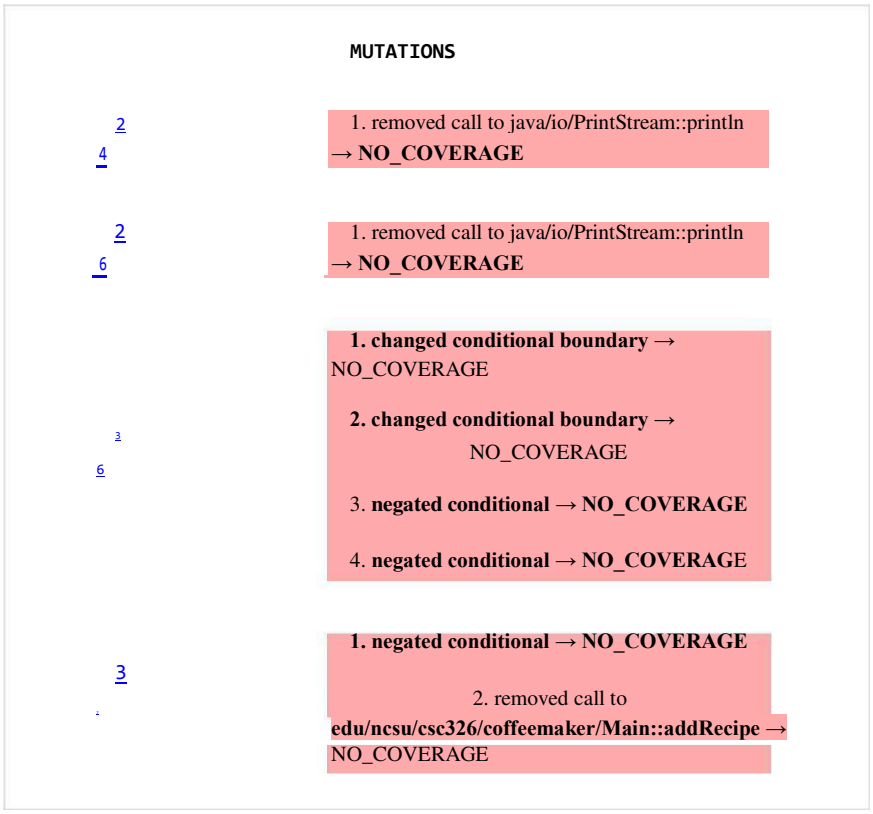
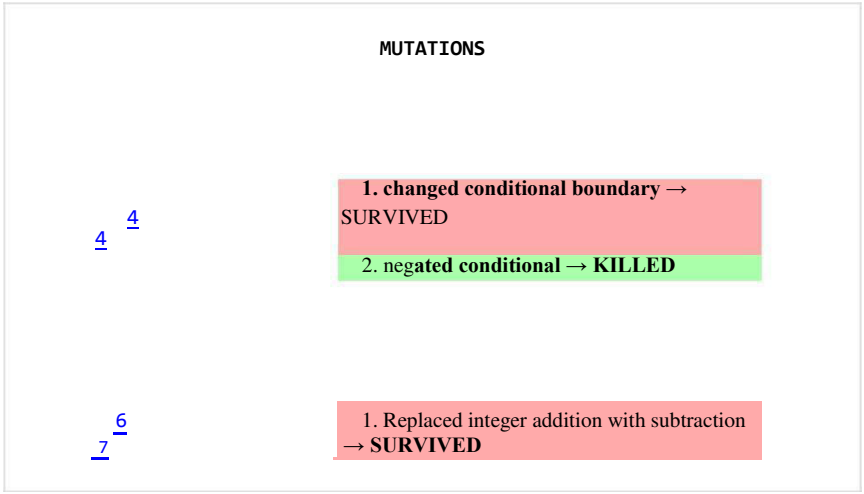


Figure 3 .Main.java mutation report.

iii. *Inventory Class Report:* Inventory.java mutation report is shown in Fig 4 where the mutation coverage is only 53%. The mutation test case for Inventory.java is not a good test cases because the mutation test case is not all killed. There is some mutation test cases survived.



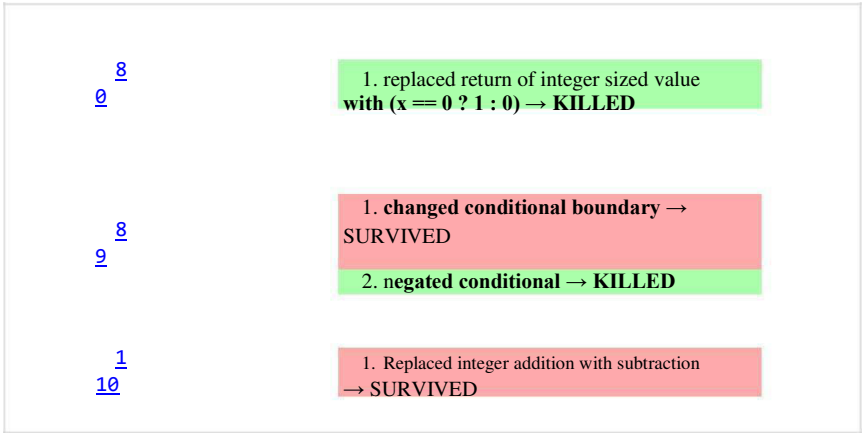


Figure 4. Inventory.java mutation report.

iv. *Recipe Class Report:* Recipe.java mutation report is shown in Fig 5 where the mutation coverage for mutation testing is 53%. It is not a very good mutation test cases for mutation testing because the mutation test cases has some of the mutation test cases survived and has no mutation coverage. Even though, some of the mutation test cases has been killed, but if there is some of the mutation test cases is still survived it is still not a very good mutation test cases for the mutation testing. In order to get good mutation test cases for the mutation testing, some changes to the mutation test cases need to be done. The first change is to make the mutation test cases has 100% line coverage. Then the mutation test cases need to be change again until the mutation coverage reach 100% of mutation coverage for the mutation testing. So, the mutation test cases can be categorizing as a good mutation test cases for the mutation testing.

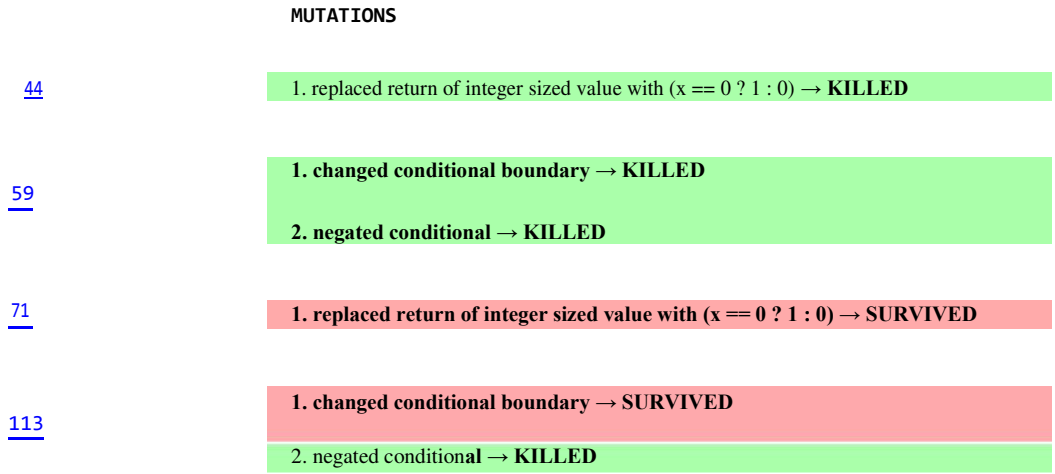
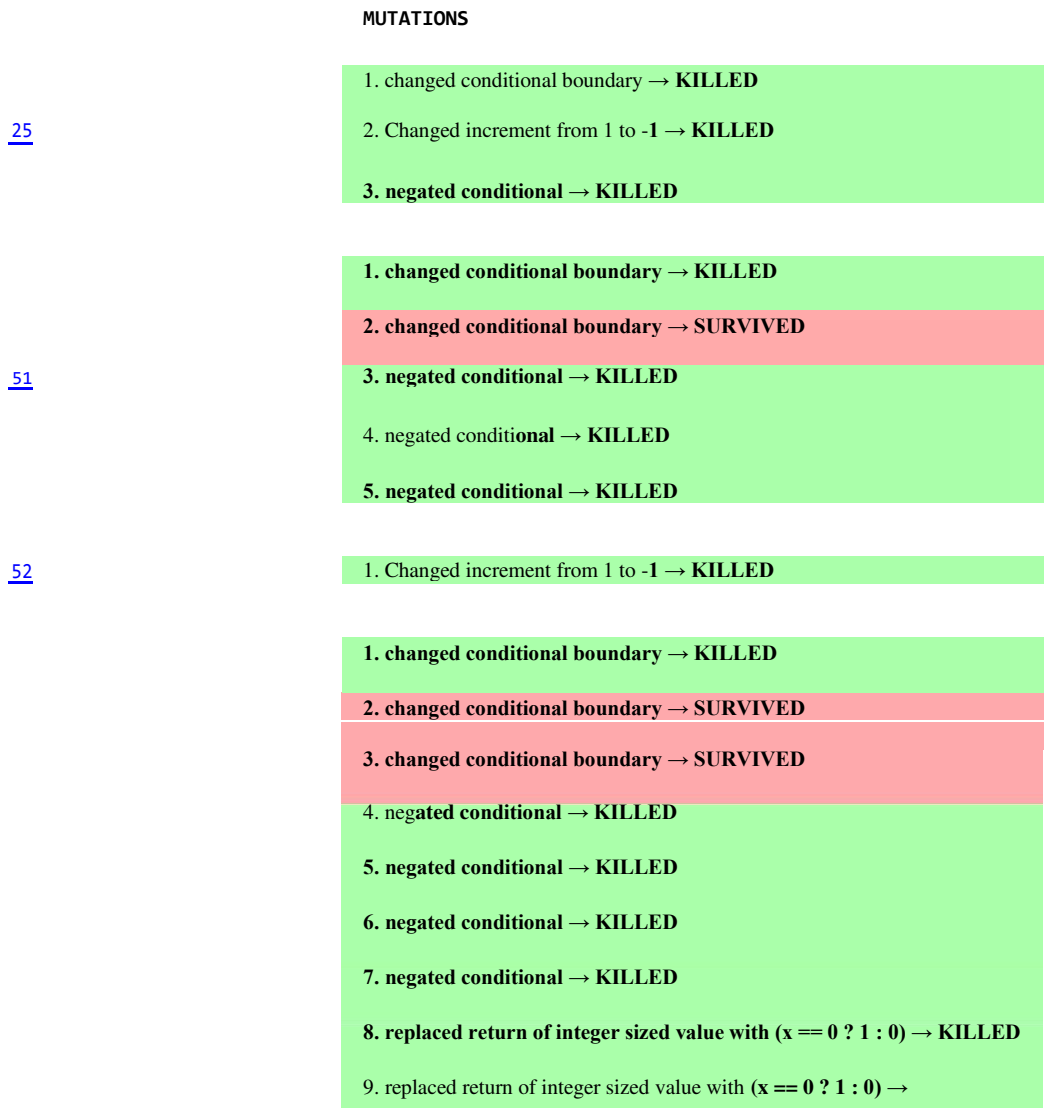


Figure 5. Recipe.java mutation report.

v. *Recipe Book Class Report:* Fig 6 shows RecipeBook.java mutation report where the mutation coverage for mutation testing is 85%. This is considering a good mutation testing but not a very good mutation testing because some of the mutation test cases are survived. The survive mutation test case need to be changes until it be killed. Before mutation coverage reach 100% coverage it is still not a good mutation testing. If want a very good mutation testing, the mutation coverage need to be 100%.



KILLED **Figure 6**. RecipeBook.java mutation report.

9. 4 Discussion

From the analysis, there are some mutation operators that cover by some mutation testing tools. Not all mutation testing tools cover all mutation operators. Mutation operators that cover by Pitest testing tools are divided into two which is mutation operators activated by default and mutation operators deactivated by default.

A. Type of mutation operators in Pitest

Mutation operators that are available for pitest is divide into two sections which is activated by default and deactivated by default. Table II will shows the list of type of mutation operators that active by default meanwhile Table III will shows the list of type of mutation n operators that deactivated by default [3]. Both tables also show the example of original condition of source code and the mutated conditional of source code.

TABLE 11. TYPE OF MUTATION OPERATORS ACTIVATE BY DEFAULT.

Mutator	Original Conditional		Mutated Conditional
Conditionals Boundary	<=		<
Increments	++		--
Invert Negatives	return -i;		return i;
Math	+		-
Negate Conditionals	==		!=
Return Values	Boolean = true		Boolean = false
Void Method Calls	public int foo() { int i = 5; return i; }		public int foo() { int i = 5; doSomething(i); return i; }

TABLE 111. TYPE OF MUTATION OPERATORS DEACTIVATED BY DEFAULT.

Mutator	Original Conditional	Mutated Conditional
Constructor Calls	Object o = new Object();	Object o = null;
Inline Constant	int i = 42;	int i = 43;
Non Void Method Calls	int i = someNonVoidMethod();	int i = 0;
Remove Conditionals	if (a == b)	if (true)

Experimental Member Variable	private final int x = 5;	private final int x = 0;

Experimental Switch	public class A {
	private static final int VAR = 13
	public String foo() {
	final int i = 42;
	return "" + VAR + ":" + i;
	}
	}
		public class A {
		public String foo() {
		return "13:42";
		}
		}

B. Mutation operators for case study.

Mutation operators that are generated for Coffee Maker case study are mutation operators that activated by default. The list of mutation operators that are generated are list in Table IV. The mutation operators that are generated in test cases for Coffee Maker case study is 247 mutation operators. From the generated mutation operators only 93 mutation operators out of 247 mutation operators that has been killed. Others mutation operators are survived and has no coverage for the mutation operators in the test cases. The mutation coverage for the mutation testing is represents in per cent. Thus, the mutation coverage for mutation testing on Coffee Maker case study is only 38 per cent. It means that our test is not as good as it should be. In order to achieve the good mutation testing, some changes in the mutation test cases need to be done until the mutation coverage reach 100% coverage.

TABLE IV. LIST OF MUTATION OPERATORS FOR COFFEE MAKER CASE STUDY.

Mutator	Generated	Killed	Survived	No Coverage	Mutation Coverage (%)
Conditionals Boundary ditionals Boundary	36	8	19	9	22
Increments	6	3	0	3	50
Void Method Calls	78	7	2	69	9
Return Values	39	28	4	7	72
Math	15	1	10	4	7
Negate	73	46	3	24	63

Conditionals					
Total	247	93	38	116	38

5 Conclusions

From the findings, the research concluded that the mutation testing for java applications have been experimented based from the research studies. Several experiments has been be done in order to get the research findings which is analyze the mutation testing result from the mutation testing on java applications.

Research constraints experienced during the completion of the research related to the topic of mutation testing on java applications is there are some obstacles that cause some types of tools had to be used in order to achieve the objectives of this research.

Obstacles encountered during the experiment are among the tools used is the mutation testing tool did not correspond to the latest version. The tools or plugin is not suitable to the Eclipse development environment. Another hurdles is the JRE(Java Runtime Environment) library support the mutation testing tools also out-dated and not the current version. Many experiments need to be done according to the version of mutation testing plugin and JRE library that suitable to the latest version of Eclipse development environment.

The research for mutation testing is not limited for java applications only. Further study can be done in the future related to this topic mutation testing for java applications but in different scope. The suggestions for improvements is the research topic can be related to this topic such as the mutation testing for mobile applications or smart watch applications as we all know this two applications is the most demanded applications by users.

Acknowledgment

In carrying out this thesis, I have been through thick and thin to explore and learn new things that I never knew. Throughout this thesis, I was able to learn something about mutation testing. Here, I want to express my gratitude to my supervisor Dr. Radziah Mohamad who has given a lot of instructions and guidance when I prepared this thesis. She gave a lot of encouragement words and support when I face with any problems during my research.

References

1. Al-Ahmad, A. S., Aljunid, S. A., & Sani, A. S. (2013). Mobile Cloud Computing Testing Review. 2013 International Conference on Advanced Computer Science Applications and Technologies. doi:10.1109/acsat.2013.42.
2. Appelt, D., Nguyen, C. D., Briand, L. C., & Alshahwan, N. (2014). Automated testing for SQL injection vulnerabilities: an input mutation approach. Proceedings of the 2014 International Symposium on Software Testing and Analysis - ISSTA 2014. doi:10.1145/2610384.2610403.
3. Available mutators. (n.d.). Retrieved June 11, 2017, from http://pitest.org/quickstart/mutators/..
4. Beginner's Guide to Mobile Application Testing. (n.d.). Retrieved June 11, 2017, from http://www.softwaretestinghelp.com/beginners-guide-to-mobile-application-testing/.
5. Deng, L., Mirzaei, N., Ammann, P., & Offutt, J. (2015). Towards mutation analysis of Android apps. 2015 IEEE Eighth International Conference on Software Testing, Verification and Validation Workshops (ICSTW). doi:10.1109/icstw.2015.7107450.
6. Hafiz, M., Ma, Y.S., & Offutt, J. (2005). Mutation Testing Tool for Java.
7. Haschemi, S., & Weißleder, S. (2010). A Generic Approach to Run Mutation Analysis. Testing – Practice and Research Techniques Lecture Notes in Computer Science, 155-164. doi:10.1007/978-3-642-15585-7_15.
8. Hsueh, M., Tsai, T., & Iyer, R. (1997). Fault injection techniques and tools. Computer, 30(4), 75- 82. doi:10.1109/2.585157.

9. Jia, Y., & Harman, M. (2011). An Analysis and Survey of the Development of Mutation Testing. IEEE Transactions on Software Engineering, 37(5), 649-678. doi:10.1109/tse.2010.62.
10. Malini, A., Venkatesh, N., Sundarakantham, K., & Mercyshalinie, S. (2014). Mobile application testing on smart devices using MTAAS framework in cloud. International Conference on Computing and Communication Technologies. doi:10.1109/iccct2.2014.7066751.
11. Ma, Y. S., & Offutt, J. (2016, July). Description of muJava's Method-level Mutation Operators. Retrieved June 11, 2017, from http://www.bing.com/cr?IG=5C8D7FBEE3464734838305063037559C&CID=1B88A48A95D960C93485AE1494DF619D&rd=1&h=nHl0W8N8Rng2SiVMk7TmdNn9FP4CoeZmjLmqu9et1Lo&v=1&r=http%3a%2f%2fcs.gmu.edu%2f%7eoffutt%2fmujava%2fmutopsMethod.pdf&p=DevEx,5061.1.
12. Mirshokraie, S., Mesbah, A., & Pattabiraman, K. (2015). Guided Mutation Testing for JavaScript Web Applications. IEEE Transactions on Software Engineering, 41(5), 429-444. doi:10.1109/tse.2014.2371458.
13. Muccini, H., Francesco, A. D., & Esposito, P. (2012). Software testing of mobile applications: Challenges and future research directions. 2012 7th International Workshop on Automation of Software Test (AST). doi:10.1109/iwast.2012.6228987
14. Nishiura, K., Maezawa, Y., Washizaki, H., & Honiden, S. (2013). Mutation analysis for Java script web application testing. In Proceedings of the International Conference on Software Engineering and Knowledge Engineering, SEKE (January ed., Vol. 2013- January, pp. 159-165). Knowledge Systems Institute Graduate School.
15. Offutt, J., Ma, Y., & Kwon, Y. (2006). The class-level mutants of MuJava. Proceedings of the 2006 international workshop on Automation of software test - AST 06. doi:10.1145/1138929.1138945
16. Pan, J. (1999). Software Testing. Retrieved June 11, 2017, from https://users.ece.cmu.edu/~koopman/des_s99/sw_testing/
17. Saifan, A. A., & Wahsheh, H. A. (2012). Mutation operators for JADE mobile agent systems. Proceedings of the 3rd International Conference on Information and Communication Systems - ICICS 12. doi:10.1145/2222444.2222460.