ONTOLOGY BASED NEGATIVE SELECTION APPROACH FOR MUTATION TESTING

SHEROLWENDY ANAK SUALIM

A thesis submitted in fulfilment of the
requirements for the award of the degree of
Master of Philosophy

School of Computing
Faculty of Engineering
Universiti Teknologi Malaysia

MAY 2019

DEDICATION

This thesis is dedicated to my beloved family and friends, who taught me that the best kind of knowledge to have is that which is learned for its own sake. It is also dedicated to my mother, who taught me that even the largest task can be accomplished if it is done one step at a time.

ACKNOWLEDGEMENT

I would like to express my appreciation to my main supervisor, Dr Radziah Mohamed for her incredible advice, encouragement, support and guidance through my Master journey. I am really proud to be under her supervision. I also want to thank my co-supervisor, Dr Nor Azizah Sa'adon for her advises and guidance in completing my thesis.

I also want to thank my family for their encouragement and moral support during this journey. I am also being thankful to my colleagues and those who have supported and provided all kinds of assistance either directly or indirectly during the completion of this projects.

Lastly, I want to express my gratitude to Universiti Teknologi Malaysia (UTM) and Ministry of Higher Education (MOHE) Malaysia for providing the facilities during this research.

ABSTRACT

Mutation testing is used to design new software tests and evaluate the quality of existing software tests. It works by seeding faults in the software program, which are called mutants. Test cases are executed on these mutants to determine if they are killed or remain alive. They remain alive because some of the mutants are syntactically different from the original, but are semantically the same. This makes it difficult for them to be identified by the test suites. Such mutants are called equivalent mutants. Many approaches have been developed by researchers to discover equivalent mutant but the results are not satisfactory. This research developed an ontology based negative selection algorithm (NSA), designed for anomalies detection and similar pattern recognition with two-class classification problem domains, either self (normal) or non-self (anomaly). In this research, an ontology was used to remove redundancies in test suites before undergoing detection process. During the process, NSA was used to detect the equivalent mutant among the test suites. Those who passed the condition set would be added to the equivalent coverage. The results were compared with previous works, and showed that the implementation of NSA in equivalent mutation testing had minimized local optimization problem in detector convergence (number of detectors) and time complexity (execution time). The findings had more equivalent mutants with average of 91.84% and scored higher mutation score (MS) with average of 80% for all the tested programs. Furthermore, the NSA had used a minimum number of detectors for higher detection of equivalent mutants with the average of 78% for all the tested programs. These results proved that the ontology based negative selection algorithm had achieved its goals to minimize local optimization problem.

ABSTRAK

Ujian mutasi digunakan untuk merekabentuk ujian perisian baru dan menilai kualiti ujian perisian yang sedia ada. Ia berfungsi dengan pembenihan kerosakan dalam program perisian yang dipanggil mutan. Kes ujian dilaksanakan pada mutan untuk menentukan jika mereka terbunuh atau masih hidup. Mutan masih hidup disebabkan oleh sesetengah mutan berbeza secara sintetik dari asal tetapi menghasikan jawapan yang sama. Ini menjadikan mereka sukar untuk dikenal pasti oleh kes ujian. Mutan tersebut dipanggil mutan bersamaan. Banyak pendekatan telah dibangunkan oleh para penyelidik untuk mencari mutan setara tetapi hasilnya tidak memuaskan. Kajian ini menghasilkan algoritma pemilihan negatif berasaskan ontologi (NSA) yang direka untuk mengesan perubahan dan pengiktirafan corak yang sama dengan domain klasifikasi masalah dua kelas, sama ada sendiri (normal) atau bukan diri (anomali). Dalam kajian ini, ontologi membantu membuang lebihan set ujian sebelum menjalani proses pengesanan. Semasa proses ini, NSA digunakan untuk mengesan mutan bersamaan dalam set ujian. Mutan yang menepati persamaan akan dimasukkan ke dalam liputan mutan bersamaan. Hasilnya dibandingkan dengan kaedah sebelumnya, dan menunjukkan bahawa pelaksanaan NSA dalam ujian mutasi bersamaan telah meminimumkan masalah tempatan pengoptimuman dalam pengesan penumpuan (jumlah pengesan) dan masa kerumitan (tempoh masa). Kajian ini telah mengesan mutan bersamaan dengan purata 91.84% dan menjaringkan skor mutasi (MS) yang lebih tinggi dengan purata 80% untuk semua program yang diuji. Tambahan pula, NSA telah menggunakan bilangan pengesan minimum untuk pengesanan mutan bersamaan yang lebih tinggi dengan purata 78% untuk semua program yang diuji. Keputusan ini membuktikan bahawa algoritma pemilihan negatif berasaskan ontologi telah mencapai matlamatnya untuk meminimumkan masalah tempatan pengoptimalan.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

ABS	-	Absolute Value Insertion
AIS	-	Artificial Immune Systems
AMC	-	Access Modifier Change
AOD	-	Arithmetic Operator Deletion
AOI	-	Arithmetic Operator Replacement
AUT	-	Application Under Test
CDL	-	Condition Deletion
COI	-	Conditional Operator Insertion
COR	-	Conditional Operator Replacement
CPU	-	Central Processing Unit
EMD	-	Equivalent Mutant Detection
FOM	-	First Order Mutation
GUI	-	Graphical User Interface
HAZOP	-	Hazard and Operability Studies
HOM	-	Higher Order Mutation
JSI	-	Static Modifier Insertion
JUnit	-	Java Unit
IHD	-	Hiding Variable Deletion
IHI	-	Hiding Variable Insertion
IOD	-	Overriding Method Deletion
IOR	-	Overriding Method Rename
ISI	-	Super Keyword Insertion
LCR	-	Logical Connector Replacement
LOC	-	Line of Code
LOI	-	Logical Operator Insertion
MS	-	Mutation Score
NSA	-	Negative Selection Algorithm
OAN	-	Argument Number Change
OO	-	Object Oriented
OWL	-	Web Ontology Language

PCC	-	Cast Type Casting
PUT	-	Program Under test
ROR	-	Relational Operator Replacement
SOM	-	Second Order Mutant
UOI	-	Unary Operator Insertion

LIST OF SYMBOLS

LIST OF APPENDICES

APPENDIX	TITLE	PAGE

CHAPTER 1

INTRODUCTION

1.1 Overview

Software testing is defined as an activity to check whether the actual results match the expected results and to ensure that the software system is defect free. It involves the execution of a software component or the system component to evaluate one or more properties of interest. Software testing also helps to identify errors, gaps or missing requirements in contrary to the actual requirements. It can be either done manually or using automated tools. In simple terms, software testing means Verification of Application Under Test (AUT). Typically testing can be classified into three categories: functional testing, non-functional testing and maintenance. In functional testing, there are several types like Unit Testing and User Acceptance Testing. While non-functional testing has types like performance and usability testing. For maintenance category, there are regression testing and maintenance.

There are several major methods used while conducting various software testing types within various software testing levels. Methods of testing include black box testing, white box testing, and experience based testing. Black box testing is a method that tested the internal structure of the program that is not known to the tester. The testing parts can be functional or non-functional, but mostly functional. Black box testing focuses on the outputs generated in response to selected inputs and execution condition (Liu and Tan, 2009).White box testing is a method that tested the internal structure of the program that is known to the tester. The test cases of this testing method were designed based on the information derived from the source code (Liu and Tan, 2009). While, agile testing is a method of the software testing that follows the principles of agile software development. Agile testing method has the ability to rapidly accommodate changes in the original requirements and prioritize the development of functionality through executable code (Collins et al., 2012).

Mutation testing is fall under the unit testing. Unit testing is a method where components or individual units of software are tested to determine their conformity to the designed specifications, and that also includes testing associated data and usage procedures. A unit is simply a small piece of code for any single function. The unit test itself is a short script or piece of code designed to verify the behavior of a particular unit to produce a pass or fail result. The aim of the unit test is to allow developers to run as many unit tests as possible to identify potential loopholes. Once the application has passed the unit testing, other forms of testing will then need to be applied for further validation. Some benefits of unit testing like faster the development as less time of debugging and reduce the cost for future maintenance. Although the benefits of Unit Testing are beginning to be understood more widely, there are still number of reasons why it has not been more fully adopted, which leaves its potential unrealized. There is no allocated time for unit testing as writing unit testing is time consuming. Besides, the emergence of new tools in the market causes difficulties to the developer to write unit tests. The more the coverage of the code, the more test coverage should be prepared by the developer.

Nowadays, testing is quite expensive because of either inefficient test suites or ineffective defect detection. In some research, mutation testing had been proven to have high cost-effective test after improving their rate of test suites and also by test cases creation where it is necessary. The idea of mutation testing is syntactically equivalence where firstly, mutation testing is used to generate mutants from original program. Each mutant contains at least one artificial change. Normally the changes are a fault in good test case. Nonetheless, sometimes the changes of the code will lead to the exhibition of the same behavior. So, in general, mutation testing achieves two goals: evaluating test suite quality and executing a test case against the original program to find errors.

Mutation testing is a type of software testing where we mutate (change) certain statements in the source code and check if the test cases are able to find the errors. It is a type of White Box Testing which is mainly used for Unit Testing. Mutation testing is one general methodology used to test program. The difference is that mutation testing focuses on the test cases used to test the program and does not

focus on the functionality of the program. The main idea of mutation testing is to create well sets test case that able to discover all the ease and hard-to-find faults in the program. So, for the application, first the original program is executed with the same test suite. Then, mutants are generated and executed with the same test suite. The execution is using a mutation operator and it checks whether the testing can identify this fault (Budd, 1981). A small syntactic modification is inserted in the program under test and it automatically created the version of the program called mutant. A mutant in the mutated program is said to be killed if any test case can distinguish the mutant from its original program. This killed mutant is behaving differently from the original program. However, if there is no test case can differentiate between them then the mutant is still alive. In order to identify equivalent mutants, the set of live mutants is analyzed. A mutant is considered equivalent if it exhibits the same deportment as the program under test. Incipient test cases are engendered to kill the live mutants. Some quandaries may occur like a high number of generated mutants, the computational cost of implementation and high effort for equivalent mutant identification even though mutation testing has benefits in term of effectiveness (Delamaro et al., 2007).

There are two general types of mutation testing, either non-equivalent or equivalent. Non-equivalent mutant is a mutant that totally differs from the original program. The difference is from the structure of the mutant that produced different output from the original. These types of mutants are easily identified during the detection process. Those who generate different output from the original are considered as a non-equivalent mutant. Equivalent mutant is defined when the introduced change does not modify the meaning of the original program. Figure 1.1 shows the example of the equivalent mutant where both codes produced the same output. A mutant is equivalent if there is no such test case that is able to differentiate between two different outputs, mutant or original program. The equivalent mutant problem is one of the most crucial problems and mutation testing widely studied over decades (Jacobs and Bean, 1963). There are several serious consequences due to the equivalent mutant. For instance, it is difficult to assess the effect of a single change of the code if there are any random changes generated. Besides, the test suite tends to find more and more non-equivalent mutant with a fixed number of equivalent mutants and percentage is statically increases (Madeyski et al., 2014). A test suite is

categorized as effective if it has the capability to detect more mutants. Mutation score used to measure the efficiency of a test suite in mutation testing. The mutation score is calculated by dividing the number of equivalent mutants detected with an overall number of non-equivalent mutants. Based on the definition of mutation score, it is assured that the detection of all equivalent mutants is very important.

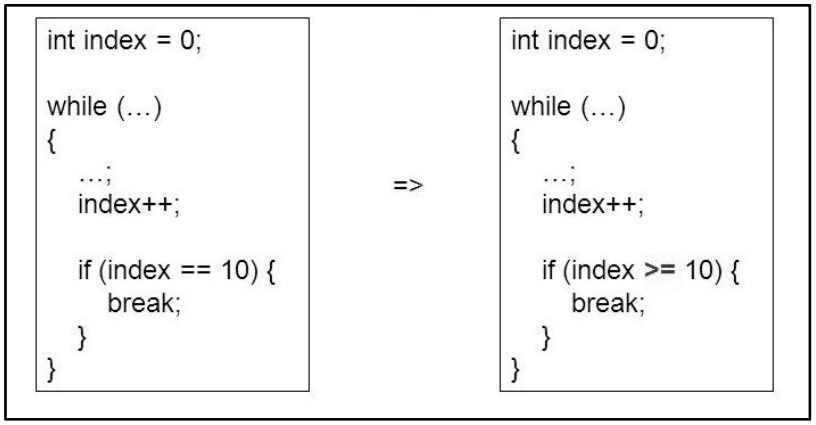

Figure 1.1The example of equivalent mutant.

The use of mutation testing is basically improving a test suite. Mutation testing provides tests for undetected mutants. After applying a mutant to a program, then one checks whether the test suite detects mutations or not. If the results show the undetected mutants, the programmer will add or modify the existing tests to detect the undetected mutants. Several reasons proved why test suite might fail to detect mutation. The reason is mutation did not change the program semantically, therefore it causes mutant to be undetected. These equivalent mutants give disadvantages to the test suite and also give additional burden to the programmer for manual assessment. If a mutation is covered but not detected, that means the test does not thoroughly check the result enough or the input data not the best for triggering erroneous behavior.

1.2 Background of the Problem

One of the major problem causes by mutation testing is the equivalent mutant problem. This problem might happen when a mutation over the original program

does not change the semantics of the program. It will be hard for the test case to detect the change. An equivalent mutant is generated due to a mutation leads to no possible observable change in behavior. This mutant is syntactically different but semantically identical to the original program. It is said to be equivalent if there is no such test case that able to differentiate between the mutant and their original output program. The result of survive mutations not found by the test suite thus mutants that are equivalent had mixes all valuable mutations in one set. This is the important issue to be considered when generating the mutants.

The equivalent mutant problem is actually a decision problem that allows the determination of a program behavioral that equivalent to its mutant. There are several reasons on the cause of equivalent mutant and their distribution to mutation testing. The reasons for a certain equivalent mutants are caused by the dead code. However, the biggest single reason is weak mutation testing cannot kill these mutants. Furthermore, those mutants that cannot be killed by strong mutation testing also become a large effect in internal state result rather than those who fail to propagate to an output.

There are several techniques that can solve the equivalent mutant problem. Some of the techniques or approaches used to solve this equivalent problem lacking in mutant coverage and time complexity. Mutant coverage includes the number of mutant detected while time complexity mentioned about the total time required by the program to run a completion. We could easily live with equivalent mutants with a few numbers or easily to be assessed. Unfortunately, it is hard to hold as there are plenty of equivalent mutants and assessing each equivalent mutant is time consuming. The effort needed to check equivalent mutant can be very high even for small programs. In fact, these mutants effectively prohibited any automatic assessment of test quality during mutation testing.

The equivalent mutant problem is a major hindrance to mutation testing. Being un-decidable in general cause is only susceptible to partial solutions. Thus, equivalent mutant has instance introduce further difficulties to the mutation testing process.

1.3 Problem Statement

The problems detected from the previous works are coverage of mutants and the execution time for testing process. Most of the existing techniques that have been tested before produce least mutant coverage and very time consuming. While these complexities of software systems have increased in recent years have required higher quality that stress on testing in order to increase quality of the testing. The importance of having highest mutant coverage is to measure the effectiveness of the method either the method able to detect high coverage of mutant.

The other problem apart from detecting high coverage of mutant is the redundancy in test suite. The redundancy causes the method to take longer time to do the testing. Due to the repetitive data in the test suite it makes the method to run the testing for same data twice. Avoiding redundancy is important for effectiveness and efficiency of the method. Previous works exclude redundancy in their testing. They directly implement the testing without avoiding any redundancy. This will affect the execution time of the testing. The execution time is important in mutation testing as this will measure the ability of the method to finish the process within shorter time.

The general research question is:

How can mutant coverage and execution time be improved using negative selection algorithm?

In order to answer this question, the following questions or challenges are required to be answered, which are:

i. How to minimize the redundancy in test suite using proposed method?
ii. How to improve mutant coverage and execution time?
iii. How effective is proposed method in solving equivalent mutant problem?

1.4 Objectives

The objectives of the research based on the problem statement are as follows:

i. To design ontology for mutation testing that identifies redundancy in test data.
ii. To apply negative selection algorithm in improving mutant coverage and time complexity.
iii. To evaluate the findings of proposed method of solving equivalent mutant

1.5 Research Scope and Significant

As described earlier, although a few studies have been done on equivalent mutant problems, but a little information was available on the process of detecting equivalent mutant in mutation testing. All of the previous researches on the related field have been encountering several problems but have yet not fully covered the solution for equivalent mutant problem. Thus, this research considered the through detection of equivalent mutant and this present as a detection technique of equivalent mutant as well as resolving the problems faced by previous approaches or methods.

The major differences between this research and previous researches can be explained as follows: first, this proposed method will be used to evaluate Java programs, second, this research will use common Java testing tools like Pitest and Muclipse, third, this research will detect equivalent mutant using its own simple and accurate rules (Euclidean distance), and last this research has a major function which is fault detection. Although NSA has been successfully applied in pattern recognition, fault detection and computer security but its applicability in the theater of operations of testing is still undiscovered. The general flow of NSA in fault detection is that a set of fault detectors are generated using NSA which is responsible for detection faulty. The self and non-self theory in NSA inspired researchers to do fault detection. This is why NSA suitable for the area of fault detection.

The main contribution of this research can be explained as follows: first, this research evolves a method that puts up an effective way for detecting equivalent mutant with the optimal set of test data that guarantees all mutants are covered, second, this research cuts the number of test data while avoiding redundancies and generated adequate test data which can be applied to detect program faults. Finally the maturation of the systematic survey includes a comparison done between the proposed method and the previous techniques. The result of this research could be useful to the solution of equivalent mutant problem. With the advantages of the negative selection approach, it could significantly increase the detection of equivalent mutant in mutation testing.

1.6 Thesis Outline

The thesis is structured as follows:

- Chapter 1: This chapter presents an introduction of the research, which includes the background of the research, problem statement, objectives and finally, the scope and significance of the study.
- Chapter 2: This chapter presents the literature review of the study. It begins with an outline of software testing, mutation testing and negative selection, as considerably as the different techniques employed to generate test data and detect equivalent mutant and discussion on related works.
- Chapter 3: This chapter presents the research methodology which includes the research operational framework, description of the data set used and declaration of the evaluation measurement.
- Chapter 4: This chapter presents the evolution of ontology for mutation testing. This includes ontology construction such as entity extraction, taxonomy formation, relationships and axioms. There is also ontology validation to determine consistency, correctness and quality.
- Chapter 5: In this chapter, it explained the experimental analysis of applying ontology based NSA in generating test data and detecting

equivalent mutant, and comparing the results with others to investigate the strength of the algorithm.

- Chapter 6: This chapter shows the validation of the proposed method. A statistical tool will be utilized to examine and compare the proposed method with other existing methods.
- Chapter 7: The contributions, conclusions as well as the suggestions for future works will be discussed in this chapter

1.7 Summary

This chapter presents the introduction of the research. The introduction is about the overview of the mutation testing including methods applied by the researchers and the problems of mutation testing that still unsatisfactory. This chapter also stated several research questions and objectives to be achieved followed by the scope and the significance of the research. And, finally, the chapter described the organization of the thesis from Chapter 1 till Chapter 7.

REFERENCES

Acree Jr, A. T. (1980). On Mutation (No. GIT-ICS-80/12). *Georgia Inst of Tech Atlanta,* School of Information And Computer Science.

Adamopoulos, K., Harman, M., and Hierons, R. M. (2004, June). How to overcome the equivalent mutant problem and achieve tailored selective mutation using co-evolution. In Genetic and evolutionary computation conference (pp. 1338-1349). Springer, Berlin, Heidelberg.

Afzal, W., Torkar, R., and Feldt, R. (2009). A systematic review of search-based testing for non-functional system properties. *Information and Software Technology*, *51*(6), 957-976.

Ahmed, M. A., and Hermadi, I. (2008). GA-based multiple paths test data generator. *Computers & Operations Research*, *35*(10), 3107-3124.

Alshahwan, N., and Harman, M. (2011, November). Automated web application testing using search based software engineering. In *Proceedings of the 2011 26th IEEE/ACM International Conference on Automated Software Engineering*(pp. 3-12). IEEE Computer Society.

Baldwin, D., and Sayward, F. (1979). *Heuristics for Determining Equivalence of Program Mutations*. *Georgia Inst of Tech Atlanta,* School of Information And Computer Science.

Black, R. (2009). Advanced Software Testing–Guide to the ISTQB Advanced Certification Vol. 1 and 2. *ISBN-13*, 978-1.

Budd, T. A. (1981). Mutation analysis: Ideas, examples, problems and prospects. *Computer Program Testing*, *8*, i29-l48.

Budd, T. A., and Angluin, D. (1982). Two notions of correctness and their relation to testing. *Acta Informatica*, *18*(1), 31-45.

Castro, L. N., De Castro, L. N., and Timmis, J. (2002). *Artificial immune systems: a new computational intelligence approach*. Springer Science & Business Media.

Cohen, M. B., Colbourn, C. J., and Ling, A. C. (2003, November). Augmenting simulated annealing to build interaction test suites. In *Software Reliability Engineering, 2003. ISSRE 2003. 14th International Symposium on* (pp. 394-405). IEEE.

Colanzi, T. E., Assunção, W. K. G., Vergilio, S. R., and Pozo, A. (2011, September). Integration test of classes and aspects with a multi-evolutionary and coupling-based approach. In *International Symposium on Search Based Software Engineering* (pp. 188-203). Springer, Berlin, Heidelberg.

Coles, H., Laurent, T., Henard, C., Papadakis, M., and Ventresque, A. (2016, July). Pit: a practical mutation testing tool for java. In *Proceedings of the 25th International Symposium on Software Testing and Analysis* (pp. 449-452). ACM.

Collins, E., Dias-Neto, A., and de Lucena Jr, V. F. (2012, July). Strategies for agile software testing automation: An industrial experience. In *Computer Software and Applications Conference Workshops (COMPSACW), 2012 IEEE 36th Annual* (pp. 440-445). IEEE.

Corcho, O., Fernández-López, M., and Gómez-Pérez, A. (2003). Methodologies, tools and languages for building ontologies. Where is their meeting point?. *Data & knowledge engineering*, *46*(1), 41-64.

Corcho, O., Fernández-López, M., Gómez-Pérez, A., and López-Cima, A. (2005). Building legal ontologies with METHONTOLOGY and WebODE. In *Law and the semantic web* (pp. 142-157). Springer, Berlin, Heidelberg.

Cristani, M., and Cuel, R. (2005). A survey on ontology creation methodologies. *International Journal on Semantic Web and Information Systems (IJSWIS)*, *1*(2), 49-69.

Dasgupta, D., Ji, Z., and Gonzalez, F. (2003, December). Artificial immune system (AIS) research in the last five years. In *The 2003 Congress on Evolutionary Computation, 2003. CEC'03.* (Vol. 1, pp. 123-130). IEEE.

Dasgupta, D., KrishnaKumar, K., Wong, D., and Berry, M. (2004, September). Negative selection algorithm for aircraft fault detection. In *International Conference on Artificial Immune Systems* (pp. 1-13). Springer, Berlin, Heidelberg.

Dasgupta, D., and González, F. (2002). An immunity-based technique to characterize intrusions in computer networks. *IEEE Transactions on evolutionary computation*, *6*(3), 281-291.

Delamaro, M., Pezze, M., Vincenzi, A. M., and Maldonado, J. C. (2001, October). Mutant operators for testing concurrent Java programs. In *Brazilian symposium on software engineering*(pp. 272-285).

Del Grosso, C., Antoniol, G., Di Penta, M., Galinier, P., and Merlo, E. (2005, June). Improving network applications security: a new heuristic to generate stress testing data. In *Proceedings of the 7th annual conference on Genetic and evolutionary computation* (pp. 1037-1043). ACM.

Deng, L., Offutt, J., and Samudio, D. (2017, July). Is Mutation Analysis Effective at Testing Android Apps?. In *Software Quality, Reliability and Security (QRS), 2017 IEEE International Conference on* (pp. 86-93). IEEE.

Derderian, K., Hierons, R. M., Harman, M., and Guo, Q. (2006). Automated unique input output sequence generation for conformance testing of FSMs. *The Computer Journal*, *49*(3), 331-344.

Do, H., Elbaum, S., and Rothermel, G. (2005). Supporting controlled experimentation with testing techniques: An infrastructure and its potential impact. *Empirical Software Engineering*, *10*(4), 405-435.

Domoney, C., Knox, M., Moreau, C., Ambrose, M., Palmer, S., Smith, P., and Swain, M. (2013). Exploiting a fast neutron mutant genetic resource in Pisum sativum (pea) for functional genomics. *Functional Plant Biology*, *40*(12), 1261-1270.

Esponda, F., Ackley, E. S., Forrest, S., and Helman, P. (2004, September). Online negative databases. In *International Conference on Artificial Immune Systems* (pp. 175-188). Springer, Berlin, Heidelberg.

Esponda, F., Forrest, S., and Helman, P. (2004). A formal framework for positive and negative detection schemes. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, *34*(1), 357-373.

Forrest, S., Perelson, A. S., Allen, L., and Cherukuri, R. (1994, May). Self-nonself discrimination in a computer. In *Research in Security and Privacy, 1994. Proceedings., 1994 IEEE Computer Society Symposium on* (pp. 202-212). Ieee.

Freitas, A. A., and Timmis, J. (2003, September). Revisiting the foundations of artificial immune systems: A problem-oriented perspective. In *International Conference on Artificial Immune Systems* (pp. 229-241). Springer, Berlin, Heidelberg.

Gligoric, M., Groce, A., Zhang, C., Sharma, R., Alipour, M. A., and Marinov, D. (2013, July). Comparing non-adequate test suites using coverage criteria. In *Proceedings of the 2013 International Symposium on Software Testing and Analysis*(pp. 302-313). ACM.

Gómez-Pérez, A., Fernández-López, M., and Corcho, O. (2006). *Ontological Engineering: with examples from the areas of Knowledge Management, e-Commerce and the Semantic Web*. Springer Science & Business Media.

Gonzalez, F., and Dasgupta, D. (2003). *A study of artificial immune systems applied to anomaly detection* (Doctoral dissertation, University of Memphis).

Gonzalez, F., Dasgupta, D., and Kozma, R. (2002, May). Combining negative selection and classification techniques for anomaly detection. In *Evolutionary Computation, 2002. CEC'02. Proceedings of the 2002 Congress on* (Vol. 1, pp. 705-710). IEEE.

Guarino, N., and Poli, R. (1993). Toward principles for the design of ontologies used for knowledge sharing. In *In Formal Ontology in Conceptual Analysis and Knowledge Representation, Kluwer Academic Publishers, in press. Substantial revision of paper presented at the International Workshop on Formal Ontology*.

Gruber, T. R. (1993). A translation approach to portable ontology specifications. *Knowledge acquisition*, *5*(2), 199-220.

Del Grosso, C., Antoniol, G., Di Penta, M., Galinier, P., and Merlo, E. (2005, June). Improving network applications security: a new heuristic to generate stress testing data. In *Proceedings of the 7th annual conference on Genetic and evolutionary computation* (pp. 1037-1043). ACM.

Delamaro, M. E., Offutt, J., and Ammann, P. (2014, March). Designing deletion mutation operators. In *Software Testing, Verification and Validation (ICST), 2014 IEEE Seventh International Conference on* (pp. 11-20). IEEE.

Derderian, K., Hierons, R. M., Harman, M., and Guo, Q. (2006). Automated unique input output sequence generation for conformance testing of FSMs. *The Computer Journal*, *49*(3), 331-344.

Du Bousquet, L., Delahaye, M., and Oriat, C. (2016, April). Applying a pairwise coverage criterion to scenario-based testing. In *Software Testing, Verification and Validation Workshops (ICSTW), 2016 IEEE Ninth International Conference on* (pp. 83-91). IEEE.

Fraser, G., and Zeller, A. (2011, March). Exploiting common object usage in test case generation. In *Software Testing, Verification and Validation (ICST), 2011 IEEE Fourth International Conference on* (pp. 80-89). IEEE.

Gligoric, M., Groce, A., Zhang, C., Sharma, R., Alipour, M. A., and Marinov, D. (2013, July). Comparing non-adequate test suites using coverage criteria. In *Proceedings of the 2013 International Symposium on Software Testing and Analysis*(pp. 302-313). ACM.

Grün, B. J., Schuler, D., and Zeller, A. (2009, April). The impact of equivalent mutants. In *Software Testing, Verification and Validation Workshops, 2009. ICSTW'09. International Conference on* (pp. 192-199). IEEE.

Hamlet, R. G. (1977). Testing programs with the aid of a compiler. *IEEE Transactions on Software engineering*, (4), 279-290.

Harman, M., Hierons, R., and Danicic, S. (2001). The relationship between program dependence and mutation analysis. In *Mutation testing for the new century* (pp. 5-13). Springer, Boston, MA.

Harman, M., Islam, F., Xie, T., and Wappler, S. (2009, March). Automated test data generation for aspect-oriented programs. In *Proceedings of the 8th ACM international conference on Aspect-oriented software development* (pp. 185-196). ACM.

Harman, M., Jia, Y., and Langdon, W. B. (2011, September). Strong higher order mutation-based test data generation. In *Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering* (pp. 212-222). ACM.

Harman, M., and McMinn, P. (2010). A theoretical and empirical study of search-based testing: Local, global, and hybrid search. *IEEE Transactions on Software Engineering*, *36*(2), 226-247.

Herzig, K., Just, S., and Zeller, A. (2013, May). It's not a bug, it's a feature: how misclassification impacts bug prediction. In *Proceedings of the 2013 international conference on software engineering* (pp. 392-401). IEEE Press.

Hierons, R., Harman, M., and Danicic, S. (1999). Using program slicing to assist in the detection of equivalent mutants. *Software Testing, Verification and Reliability*, *9*(4), 233-262.

Huang, T., Li, W., and Yang, C. (2008, December). Comparison of Ontology Reasoners: Racer, Pellet, Fact++. In *AGU Fall Meeting Abstracts*.

Inozemtseva, L., and Holmes, R. (2014, May). Coverage is not strongly correlated with test suite effectiveness. In *Proceedings of the 36th International Conference on Software Engineering* (pp. 435-445). ACM.

Jacobs, I. S. (1963). Fine particles, thin films and exchange anisotropy. *Magnetism*, 271-350.

Jalote, P. (2008). *A concise introduction to software engineering*. Springer Science & Business Media.

Jones, B. F., Eyres, D. E., and Sthamer, H. H. (1998). A strategy for using genetic algorithms to automate branch and fault-based testing. *the computer journal*, *41*(2), 98-107.

Jia, Y., and Harman, M. (2008, September). Constructing subtle faults using higher order mutation testing. In *Source Code Analysis and Manipulation, 2008 Eighth IEEE International Working Conference on* (pp. 249-258). IEEE.

Jia, Y., and Harman, M. (2009). Higher order mutation testing. *Information and Software Technology*, *51*(10), 1379-1393.

Jia, Y., and Harman, M. (2011). An analysis and survey of the development of mutation testing. *IEEE transactions on software engineering*, *37*(5), 649-678.

Just, R., Jalali, D., Inozemtseva, L., Ernst, M. D., Holmes, R., and Fraser, G. (2014, November). Are mutants a valid substitute for real faults in software testing?. In *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering* (pp. 654-665). ACM.

Just, R., Kapfhammer, G. M., and Schweiggert, F. (2012, April). Do redundant mutants affect the effectiveness and efficiency of mutation analysis?. In *Software testing, verification and validation (ICST), 2012 IEEE fifth international conference on*(pp. 720-725). IEEE.

Just, R., Kapfhammer, G. M., and Schweiggert, F. (2012, November). Using non-redundant mutation operators and test suite prioritization to achieve efficient and scalable mutation analysis. In *Software Reliability Engineering (ISSRE), 2012 IEEE 23rd International Symposium on* (pp. 11-20). IEEE.

Just, R., and Schweiggert, F. (2015). Higher accuracy and lower run time: efficient mutation analysis using non-redundant mutation operators. *Software Testing, Verification and Reliability*, *25*(5-7), 490-507.

Kaminski, G., Ammann, P., and Offutt, J. (2011, May). Better predicate testing. In *Proceedings of the 6th International Workshop on Automation of Software Test* (pp. 57-63). ACM.

Kaminski, G., Ammann, P., and Offutt, J. (2013). Improving logic-based testing. *Journal of Systems and Software*, *86*(8), 2002-2012.

Khan, M. E. (2011). Different approaches to black box testing technique for finding errors. *International Journal of Software Engineering & Applications*, *2*(4), 31.

Kim, J., and Bentley, P. (1999, July). Negative selection and niching by an artificial immune system for network intrusion detection. In *Proc. of GECCO'99* (pp. 149-158).

Kim, J., and Bentley, P. J. (2001, July). An evaluation of negative selection in an artificial immune system for network intrusion detection. In *Proceedings of the 3rd Annual Conference on Genetic and Evolutionary Computation* (pp. 1330-1337). Morgan Kaufmann Publishers Inc..

Kintis, M., Papadakis, M., and Malevris, N. (2015). Employing second-order mutation for isolating first-order equivalent mutants. *Software Testing, Verification and Reliability*, *25*(5-7), 508-535.

Klischies, D., and Fögen, K. (2016). An analysis of current mutation testing techniques applied to real world examples. *Full-scale Software Engineering/Current Trends in Release Engineering*, *13*.

Langdon, W. B., Harman, M., and Jia, Y. (2009, September). Multi objective higher order mutation testing with genetic programming. In *Testing: Academic and Industrial Conference-Practice and Research Techniques, 2009. TAIC PART'09.* (pp. 21-29). IEEE.

Langdon, W. B., Harman, M., and Jia, Y. (2010). Efficient multi-objective higher order mutation testing with genetic programming. *Journal of systems and Software*, *83*(12), 2416-2430.

Laurent, T., Papadakis, M., Kintis, M., Henard, C., Le Traon, Y., and Ventresque, A. (2017, March). Assessing and improving the mutation testing practice of PIT. In *Software Testing, Verification and Validation (ICST), 2017 IEEE International Conference on* (pp. 430-435). IEEE.

Lewis, W. E. (2000). *Software testing and continuous quality improvement*. Auerbach publications.

Liu, H., and Tan, H. B. K. (2009). Covering code behavior on input validation in functional testing. *Information and Software Technology*, *51*(2), 546-553.

López-Pellicer, F. J., Vilches-Blázquez, L. M., Nogueras-Iso, J., Corcho, Ó., Bernabé, M. A., and Rodríguez, A. F. (2008). Using a hybrid approach for the development of an ontology in the hydrographical domain.

Ma, Y. S., Offutt, J., and Kwon, Y. R. (2005). MuJava: an automated class mutation system. *Software Testing, Verification and Reliability*, *15*(2), 97-133.

Mabrook, A. Z. M. S. (2013). Study of software testing and the evolution of optimal method for quality and reliability investigation.

Madeyski, L., and Radyk, N. (2010). Judy-a mutation testing tool for Java. *IET software*, *4*(1), 32-42.

Madeyski, L., Orzeszyna, W., Torkar, R., and Jozala, M. (2014). Overcoming the equivalent mutant problem: A systematic literature review and a comparative experiment of second order mutation. *IEEE Transactions on Software Engineering*, *40*(1), 23-42.

Mala, D. J., Elizabeth, S. R., and Mohan, V. (2008). Intelligent Test Case Optimizer-An automated Hybrid Genetic Algorithm based test case optimization framework. *International Journal Of Computer Science And Applications*, *1*(1), 51-55.

Malhotra, R., and Garg, M. (2011). An adequacy based test data generation technique using genetic algorithms. *Journal of information processing systems*, *7*(2), 363-384.

Mathur, A. P. (1991, September). Performance, effectiveness, and reliability issues in software testing. In *1991 The Fifteenth Annual International Computer Software & Applications Conference* (pp. 604-605). IEEE.

McMinn, P., Harman, M., Lakhotia, K., Hassoun, Y., and Wegener, J. (2012). Input domain reduction through irrelevant variable removal and its effect on local, global, and hybrid search-based structural test data generation. *IEEE Transactions on Software Engineering*, *38*(2), 453-477.

McMinn, P., Shahbaz, M., and Stevenson, M. (2012, April). Search-based test input generation for string data types using the results of web queries. In *Software Testing, Verification and Validation (ICST), 2012 IEEE Fifth International Conference on* (pp. 141-150). IEEE.

Michael, C. C., McGraw, G., and Schatz, M. A. (2001). Generating software test data by evolution. *IEEE transactions on software engineering*, (12), 1085-1110.

Myers, G. J., Sandler, C., and Badgett, T. (2011). *The art of software testing*. John Wiley & Sons.

Myers, G. J., Sandler, C., Badgett, T., and Thomas, T. M. (2004). The Art of Software Testing. Business Data Processing: a Wiley Series.

Naik, K., and Tripathy, P. (2011). *Software testing and quality assurance: theory and practice*. John Wiley & Sons.

Nguyen, Q. V., and Madeyski, L. (2015). Searching for strongly subsuming higher order mutants by applying multi-objective optimization algorithm. In *Advanced Computational Methods for Knowledge Engineering* (pp. 391-402). Springer, Cham.

Nguyen, Q. V., and Madeyski, L. (2016). Higher order mutation testing to drive development of new test cases: An empirical comparison of three strategies. In *Intelligent Information and Database Systems* (pp. 235-244). Springer, Berlin, Heidelberg.

Nguyen, C. D., Miles, S., Perini, A., Tonella, P., Harman, M., and Luck, M. (2012). Evolutionary testing of autonomous software agents. *Autonomous Agents and Multi-Agent Systems*, *25*(2), 260-283.

Norouzi, M., Fleet, D. J., and Salakhutdinov, R. R. (2012). Hamming distance metric learning. In *Advances in neural information processing systems* (pp. 1061-1069).

Noy, N. F., Sintek, M., Decker, S., Crubézy, M., Fergerson, R. W., and Musen, M. A. (2001). Creating semantic web contents with protege-2000. *IEEE intelligent systems*, *16*(2), 60-71.

Offutt, A. J. (1992). Investigations of the software testing coupling effect. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, *1*(1), 5-20.

Offutt, J., and Ammann, P. (2008). *Introduction to software testing* (p. 27). Cambridge: Cambridge University Press.

Offutt, A. J., and Craft, W. M. (1994). Using compiler optimization techniques to detect equivalent mutants. *Software Testing, Verification and Reliability*, *4*(3), 131-154.

Offutt, A. J., Lee, A., Rothermel, G., Untch, R. H., and Zapf, C. (1996). An experimental determination of sufficient mutant operators. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, *5*(2), 99-118.

Offutt, A. J., and Pan, J. (1996, June). Detecting equivalent mutants and the feasible path problem. In *Computer Assurance, 1996. COMPASS'96, Systems Integrity. Software Safety. Process Security. Proceedings of the Eleventh Annual Conference on* (pp. 224-236). IEEE.

Offutt, A. J., and Pan, J. (1997). Automatically detecting equivalent mutants and infeasible paths. *Software testing, verification and reliability*, *7*(3), 165-192.

Offutt, A. J., Rothermel, G., and Zapf, C. (1993, May). An experimental evaluation of selective mutation. In *Proceedings of the 15th international conference on Software Engineering*(pp. 100-107). IEEE Computer Society Press.

Offutt, A. J., Voas, J., and Payne, J. (1996). *Mutation operators for Ada*. Technical Report ISSE-TR-96-09, Information and Software Systems Engineering, George Mason University.

Pan, J. (1994). *Using constraints to detect equivalent mutants*(Master's thesis, George Mason University).

Papadakis, M., Jia, Y., Harman, M., and Le Traon, Y. (2015, May). Trivial compiler equivalence: A large scale empirical study of a simple, fast and effective equivalent mutant detection technique. In *Proceedings of the 37th International Conference on Software Engineering-Volume 1* (pp. 936-946). IEEE Press.

Pezzè, M., and Young, M. (2008). Teste e Análise de software. *Tradução Bernardo Copstein, Flavio Moreira de Oliveira. Porto Alegre: Bookman*.

Polo, M., Piattini, M., and García-Rodríguez, I. (2009). Decreasing the cost of mutation testing with second-order mutants. *Software Testing, Verification and Reliability*, *19*(2), 111-131.

Sahaf, Z., Garousi, V., Pfahl, D., Irving, R., and Amannejad, Y. (2014, May). When to automate software testing? decision support based on system dynamics: an industrial case study. In *Proceedings of the 2014 International Conference on Software and System Process* (pp. 149-158). ACM.

Sahu, A., and Maharana, P. (2013). Negative Selection Method for Virus Detection in a Cloud. *International Journal of Computer Science and Information Technologies*, *4*, 771-774.

Schuler, D., Dallmeier, V., and Zeller, A. (2009, July). Efficient mutation testing by checking invariant violations. In *Proceedings of the eighteenth international symposium on Software testing and analysis* (pp. 69-80). ACM.

Schuler, D., and Zeller, A. (2009, August). Javalanche: efficient mutation testing for Java. In *Proceedings of the the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering* (pp. 297-298). ACM.

Schuler, D., and Zeller, A. (2010, April). (Un-) covering equivalent mutants. In *Software Testing, Verification and Validation (ICST), 2010 Third International Conference on* (pp. 45-54). IEEE.

Schuler, D., and Zeller, A. (2013). Covering and uncovering equivalent mutants. *Software Testing, Verification and Reliability*, *23*(5), 353-374.

Shelton, W., Li, N., Ammann, P., and Offutt, J. (2012, April). Adding criteria-based tests to test driven development. In *Software Testing, Verification and Validation (ICST), 2012 IEEE Fifth International Conference on* (pp. 878-886). IEEE.

Siami Namin, A., Andrews, J. H., and Murdoch, D. J. (2008, May). Sufficient mutation operators for measuring test effectiveness. In *Proceedings of the 30th international conference on Software engineering* (pp. 351-360). ACM.

Singh, O., Kapur, P. K., and Anand, A. (2012). A multi-attribute approach for release time and reliability trend analysis of a software. *International Journal of System Assurance Engineering and Management*, *3*(3), 246-254.

Smith, B. H., and Williams, L. (2007, September). An empirical evaluation of the MuJava mutation operators. In *Testing: academic and industrial conference practice and research techniques-MUTATION, 2007. TAICPART-MUTATION 2007*(pp. 193-202). IEEE.

Srivastava, P. R., and Kim, T. H. (2009). Application of genetic algorithm in software testing. *International Journal of software Engineering and its Applications*, *3*(4), 87-96.

Sommerville, I. (2011). Software Engineering, Boston, Massachusetts: Pearson Education.

Tan, R. P., and Edwards, S. H. (2004). Experiences evaluating the effectiveness of JML-JUnit testing. *ACM SIGSOFT Software Engineering Notes*, *29*(5), 1-4.

Tonella, P. (2004, July). Evolutionary testing of classes. In *ACM SIGSOFT Software Engineering Notes* (Vol. 29, No. 4, pp. 119-128). ACM.

Umar, M. (2006). An evaluation of mutation operators for equivalent mutants. *Project report, MSc in Advanced Software Engineering, Department of Computer Science, King's College London, London, UK*.

Vanoverberghe, D., de Halleux, J., Tillmann, N., and Piessens, F. (2012, January). State coverage: Software validation metrics beyond code coverage. In *International Conference on Current Trends in Theory and Practice of Computer Science*(pp. 542-553). Springer, Berlin, Heidelberg.

Voas, J. M., and McGraw, G. (1997). *Software fault injection: inoculating programs against errors*. John Wiley & Sons, Inc..

Walcott, K. R., Soffa, M. L., Kapfhammer, G. M., and Roos, R. S. (2006, July). Timeaware test suite prioritization. In *Proceedings of the 2006 international symposium on Software testing and analysis* (pp. 1-12). ACM.

Wegener, J., and Bühler, O. (2004, June). Evaluation of different fitness functions for the evolutionary testing of an autonomous parking system. In *Genetic and Evolutionary Computation Conference* (pp. 1400-1412). Springer, Berlin, Heidelberg.

Wegener, J., and Grochtmann, M. (1998). Verifying timing constraints of real-time systems by means of evolutionary testing. *Real-Time Systems*, *15*(3), 275-298.

Weyuker, E. J. (1979). Translatability and decidability questions for restricted classes of program schemas. *SIAM Journal on Computing*, *8*(4), 587-598.

Whittaker, J. A. (2000). What is software testing? And why is it so hard?. *IEEE software*, *17*(1), 70-79.

Wohlin, C., Runeson, P., Höst, M., Ohlsson, M. C., Regnell, B., and Wesslén, A. (2012). *Experimentation in software engineering*. Springer Science & Business Media.

Wright, C. J., Kapfhammer, G. M., and McMinn, P. (2014, October). The impact of equivalent, redundant and quasi mutants on database schema mutation analysis. In *Quality Software (QSIC), 2014 14th International Conference on* (pp. 57-66). IEEE.

Yoo, S., Harman, M., Tonella, P., and Susi, A. (2009, July). Clustering test cases to achieve effective and scalable prioritisation incorporating expert knowledge. In *Proceedings of the eighteenth international symposium on Software testing and analysis* (pp. 201-212). ACM.

Zhan, Y., and Clark, J. A. (2005, June). Search-based mutation testing for simulink models. In *Proceedings of the 7th annual conference on Genetic and evolutionary computation* (pp. 1061-1068). ACM.