TEST CASES REDUCTION USING SIMILARITY RELATION AND CONDITIONAL ENTROPY

NOOR FARDZILAWATI BINTI MD NASIR

A thesis submitted in
fulfillment of the requirement for the award of the
Degree of Master of Information Technology

Faculty of Computer Science and Information Technology
Universiti Tun Hussein Onn Malaysia

DECEMBER, 2017

DEDICATION

In the name of Allah, Most Gracious, The Most Merciful.

Special dedicated to my parent, Md Nasir Bin Md Isa and Jamaliah Binti Abdul Majid, my husband Shaid Bin Jaffar, my son Hallaj Fansuri Bin Shaid, my siblings and all my friends. Thanks you for your love, prayers, sopport and encouragement since this journey started until the final stages. Without you, I would not be able to go so far as where I am now. Hopefully, Allah will bless us all. Ameen.

ACKNOWLEDGEMENT

First and foremost, I would like to express my sincere gratitude and appreciation to my supervisor Dr. Noraini Binti Ibrahim who gave me the opportunity to be her student. Her guidance helps me in all the time of research and writing of this thesis.

My sincere thanks also goes to all postgraduate members, fellow experts, researchers and staff in Faculty of Computer Science and Information Technology, UTHM for their encouragement and motivation in making me take my first step into Master thesis.

I owe my loving thanks to my husband Shaid Bin Jaffar and my kid Hallaj Fansuri. Thanks to my loving parents Md Nasir Bin Md Isa and Jamaliah Binti Abdul Majid. Without their encouragement and understanding, it would have been impossible for me to finish this work. My special gratitude is due to my brothers and my sisters for their loving support.

Lastly, I also wish to thank all members that I cannot specify here for their support, cooperation and contribution along the way. Thank you to Ministry of Higher Education for sponsoring this research through Fundamental Research Grant Scheme (FRGS) vote number 1610 and Universiti Tun Hussein Onn Malaysia (UTHM).

ABSTRACT

Testing is essential in software engineering due to the huge size and complexity of everyday software systems. Generation of effective test cases becomes a crucial task due to the increment in the source code size and rapid change of requirements. New test cases are generated and added to a test suite to exercise the latest modification to software. Therefore, it is not easy to select effective test cases due to their redundancy and having common requirements. Thus, new challenges arise in reducing redundant test cases and finding common requirements that would decrease the cost and maintenance of a software testing process. Given a test suite and a set of requirements covered by test suite, Test Case Reduction or Minimization aims to select a subset of test cases that covers the same requirements. Several techniques have been proposed by researchers based on reduction parameter such as Test Suite Reduction, Fault Capability Detection and Processing Time Reduction. Nonetheless, these techniques are unable to tackle all parameters simultaneously, for example, some techniques may perform well in reducing the size of test cases but less considered on fault detection ability. To address this issue, this study proposed a technique that is able to minimize the size of test cases and common requirement attributes without compromising on fault detection capability. The proposed technique uses Similarity Relation to reduce the size of the test cases and Conditional Entropy to reduce the number of common requirements. The experimental results show a test case reduction that is smaller in size without affecting the decision of the testing. The proposed technique was able to reduce up to 50% of the reduction rate compared to base-line techniques such as MFTS Algorithm, FLOWER, RZOLTAR and Weighted Greedy Algorithm.

ABSTRAK

Saiz dan kerumitan sistem perisian yang semakin meningkat dalam kehidupan seharian menjadikan pengujian penting dalam kejuruteraan perisian. Keberkesanan kes ujian *(test case)* yang dihasilkan adalah penting disebabkan oleh peningkatan saiz kod sumber dan perubahan pesat keperluan. Kes ujian baharu dihasilkan dan ditambah ke suite ujian *(test suite)* untuk menjalankan pengubahsuaian terkini kepada perisian. Proses untuk memilih kes ujian yang berkesan adalah tidak mudah disebabkan oleh kes ujian yang lewah dan jenis keperluan yang serupa. Oleh itu, timbul cabaran baharu untuk mengurangkan kes ujian yang tidak perlu dan mencari keperluan serupa yang akan mengurangkan kos dan memudahkan proses penyelenggaraan ujian perisian. Untuk suite ujian dan set keperluan yang diliputi oleh suite ujian, Pengurangan Kes Ujian *(Test Case Reduction or Minimization)* bertujuan untuk memilih subset kes ujian yang memenuhi keperluan yang sama. Beberapa teknik telah dicadangkan oleh para penyelidik berdasarkan parameter pengurangan seperti Ujian Pengurangan Suite, Pengesanan Kemampuan Kesalahan dan Pengurangan Masa Pemprosesan. Walau bagaimanapun, teknik tersebut tidak dapat menangani semua parameter secara serentak, contohnya beberapa teknik tersebut dapat mengurangkan kes ujian dengan berkesan namun kurang mempertimbangkan keupayaan untuk mengesan kesalahan. Untuk menangani isu ini, kajian ini mencadangkan sebuah teknik yang berupaya meminimumkan saiz kes ujian dan sifat keperluan tanpa mengorbankan keupayaan pengesanan kesalahan. Teknik yang dicadangkan menggunakan Hubungan Kesamaan *(Similarity Relation)* untuk mengurangkan saiz kes ujian dan Entropy Bersyarat *(Conditional Entropy)* untuk mendapatkan subset keperluan serupa. Hasil eksperimen menunjukkan pengurangan kes ujian yang lebih kecil tanpa mempengaruhi keputusan ujian. Teknik yang dicadangkan telah berupaya mengurangkan sehingga 50% kadar pengurangan kes ujian, berbanding teknik biasa seperti Algoritma *MFTS*, *FLOWER*, *RZOLTAR* dan Algoritma Serat Berat *(Weighted Greedy Algorithm)*.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ALGORITHMS

LIST OF SYMBOLS AND ABBREVIATIONS

SDLC	-	Software Development Life Cycle
NP	-	Non deterministic Polynomial time
IS	-	Information System
ST	-	Software Testing
TCR	-	Test Case Requirement Matrix
UML	-	Unified Modeling Language
GUI	-	Graphic User Interface
HGS	-	Harrold, Gupta & Soffa algorithm
ILP	-	Integer Linear Programming
ATMS	-	Automatic Teller Machine System Independent Verification and Validation Project
MSTB	-	Malaysian Software Testing Board
GE	-	Greedy Heuristic
GRE	-	Greedy Heuristic Essential
MFTS	-	Maximal Frequent Test Set
FDC	-	Fault Detection Capability
TSR	-	Test Suite Reduction
TS	-	Test Suite
MC/DC	-	Modified Condition/ Decision Coverage
WS	-	Weighted Set
WSC	-	Weighted Set Coverage
STS	-	Set of Test Suite
RTB	-	Reduction with tie breaking
SUT	-	System under test
BOG	-	Bi objective Greedy Algorithm
TSSR	-	Test Suite Reduction Rate

TCSR	-	Test case size reduction
CRR	-	Common Requirement Reduction
RSR	-	Reduce with selective redundancy
CRSR	-	Common Requirement Size Reduction
TCSR	-	Test Case Size Reduction
SRS	-	Software Requirement and Specification
IEEE	-	Institute of Electrical and Electronic
SQA	-	Software Quality Assurance
SWEBOK	-	Software Engineering Body of Knowledge
TS	-	Test Suite
STS	-	Set of Test Suite

LIST OF APPENDICES

LIST OF PUBLICATIONS

Proceedings:

(i) **Noor Fardzilawati Md Nasir,** Noraini Ibrahim, Tutut Herawan (2016). "Detection of Redundancy in CFG-Based Test Cases Using Entropy". The Second International Conference on Soft Computing and Data Mining (SCDM-2016) (pp. 244-252). SCOPUS indexed.

CHAPTER 1

INTRODUCTION

Software development process is not limited solely to the code writing process. The requirements of the software must be defined so that the specifications of the software can be created, codes can be written and tests can be performed to ensure that the software developed matches with the intended specifications, defects are eliminated and the software can be maintained over time (Hooda & Chhillar, 2014). All these stages are known as *software development lifecycle* (SDLC). Furthermore, the developed software will also change over time due to the different strategies employed by software developers in carrying out the SDLC process. Consequently, software testing becomes an important stage and is needed throughout SDLC.

Software testing is an activity conducted by software testers to validate whether a system is working correctly or not. Nowadays, with the evolution in software system, it is a challenge for software testing mechanisms to determine whether the system has passed or failed a test, which is also a subject of interest for many researchers (Barr *et al.*, 2015). In software testing, the testing requirements are gathered from software requirement and specification (SRS) and once a set of requirements is found, a set of test cases (test suite) are generated to fulfill the requirements (DeMilli & Offutt, 1991). The development team usually has a well-specified set of test cases that should be run within minimal time. According to IEEE definition (Binkley, 1997), *test case* is a simple medium containing the inputs and expected result developed to test the program with the requirements given. *Test suite* is a collection of test cases that are grouped and run together. The generated test cases are used to ensure that the requirements gathered from SRS are satisfied by the

system (Gupta & Soffa, 1993). It is important to produce good quality test cases. A quality test case should have the properties of its overall running time, size and its expected fault detection capability or a combination of those (Khalilian & Parsa, 2009; Rothermel *et al.*, 2002).

However, during SDLC, some modifications in maintenance phase may lead to growth in software size and results to an increment in test suite size. Over time, test cases designed specifically for one requirement may satisfy additional requirement as well. Two test cases are redundant if they satisfy the same requirements (Mohapatra & Prasad, 2015; Chen & Lau, 1998). Thus, the created test suite may contain redundancy because some of its proper subsets may still satisfy the set of requirements. In some cases, some requirements are also common, with respect to any test case. For example, in a web application, 1000 test cases in a test suite are used to test 500 requirements. Some of the test cases are redundant when a specific requirement exercised by a test case is also exercised by another test case in the test suite. All these issues create the motivation to create a good technique on selecting the minimal subset of test cases that covers all requirements without hampering the decision of pass or fail of the system.

It is advantageous to have the smallest possible set of test cases because tests must be run repeatedly for every change done in the software due to the removal of redundant test cases (Mohapatra & Prasad, 2015; Chen & Lau, 1998). This process is known as test suite reduction or test suite minimization (Harris & Raju, 2015; Khalilian & Parsa, 2009; Lin & Huang, 2009; Harrold *et al.*, 1993). Both terms will be used interchangeably throughout this research. Test case reduction is a technique that will decrease the size of the test cases and time needed for test suite execution while providing the same software coverage as the original test suite (Alian *et al.*, 2016). In this technique, only necessary subsets of test cases are selected to be the representative test suite (Harrold *et al.*, 1993). In order to solve this problem, several reduction techniques have been introduced. They can be classified into: (1) *Requirement Base* that satisfies all the optimized requirements with minimum number of test cases; (2) *Genetic Algorithm* that uses computational intelligence based approach to cater the problem of evolutionary computation in test case reduction; (3) *Fuzzy Logic* that is used in the communications, bioinformatics and expert systems. This type of technique attempts to reduce test cases based on objectives function and is quite similar to human judgment; (4) *Coverage Based* that

ensures the majority of the execution paths of the software are exercised; (5) *Program Slicing* that helps to show the control flow of a software and specify which statements are invoked with that test cases; (6) *Greedy Algorithm* or heuristic for code-based reduction which selects the test cases with the maximum number of unsatisfied requirements and arbitrary choice for the tie situation; (7) *Hybrid Algorithm* which is a combination of several *genetic algorithms* that provides significant reduction of test cases and multi-objectives optimization; and (8) *Clustering* that uses the clustering technique to run the test with clustered test cases rather than with the entire test suite (Alian *et al.*, 2016).

1.1 Research Motivation

Test case generation is the most challenging part in software testing (Singh, 2014; Zeng & Tan, 2012). As software evolves, new test cases are generated and added to a test suite to exercise the latest modifications to the software. Due to many versions of the software developed, the possibility to generate redundant test cases in the test suite will increase (Singh *et al.*, 2011). Apart from that, the redundant test cases must exercise software requirements for which they were generated. There are some requirements that are common in any test case. These common requirements can be reduced without affecting system performance. All these issues create the motivation to create a good technique for selecting a minimal subset of test cases that covers all requirements without hampering the decision of pass or fail of the system. This is a NP-complete problem and can be easily proven using a polynomial time reduction from the *minimum set-cover* problem (Garey MR, 1979). The set cover problem consists of finite set of test cases, *T* and *m* subsets of requirements R_1, R_2, R_m of these test cases. The minimum set-cover problem is used to find the fewest number of test cases subsets and requirements. Therefore, the heuristics part of solving this problem is important in order to keep the test suite as small as possible while preserving the test suite quality.

Many researchers (Harris & Raju, 2015; Gotlieb & Marijan, 2014; Campos & Abreu, 2013; Xu *et al.*, 2012) have proposed various test case reduction techniques to approximate a minimal subset of test cases. MFTS Algorithm (Harris & Raju, 2015) solves this problem by optimizing the test suite based on related testing

objective. (Gotlieb & Marijan, 2014) introduced FLOWER that computes a minimum-sized test suite while preserving the coverage of requirements. RZOLTAR heuristic approach (Campos & Abreu, 2013) attempts to obtain an optimal representative set of test cases. Another research (Xu *et al.*, 2012) presents a modified greedy algorithm solution by means of Weighted Set Covering (WSC). However, the researchers aim was to reduce the number of test suite, but less attention was given to maintain the method's fault capability. Hence, there is a need for a technique that produces the minimal subset of test cases and common requirements, while simultaneously maintains the ability to detect faults. At the same time, the test cases and common requirements size can be reduced, thus reducing the time needed to run the test.

Rough Set Theory has been used for attribute selection in *Incomplete Information Systems* with significant success (Deris *et al.*, 2015; Wang, 2002). This method gives minimal reduction in incomplete decision system. Capitalizing on its advantage in handling flexible and precise data selection in *Information System* (IS), Rough Set Theory was utilized for *Software Testing* (ST). While IS focuses on finding the attribute which is low in similarity and certainty relation, ST needs minimal similarity relation among the test cases to minimize the redundancy between the test cases. This further motivates the researcher as it seemed to be a very attractive solution for the test case reduction issue.

This research applies similarity relation to classify the test cases in order to eliminate the redundant test cases. The Rough Set Theory provides the ability to find similarity relation among the test cases and reduce redundant test cases without compromising the fault detection capability of the test. Conditional Entropy is introduced to the similarity relation approach to reduce a certain number of the common requirement attributes. The goal of integrating Rough Set Theory and information theory in software testing is to seek a practical approach in software testing. However, this approach is less used by researchers in ST, especially in test cases reduction. Further research should be done to provide better results in test cases reduction.

1.2 Research Objectives

Objectives of the research are as follows:

(i) To develop Test Case Reduction technique using Similarity Relation and Conditional Entropy.

(ii) To validate and compare the proposed technique with other techniques namely MFTS Algorithm, FLOWER, RZOLTAR and Weighted Greedy Algorithm.

1.3 Scope and Limitation

This research focused only on reducing the test cases and common requirements in Test Case-Requirement (TCR) Matrix table. The relationship between test cases and associated requirements will be represented as a Test Case-Requirement (TCR) Matrix table. Similarity relation will be used to reduce the test cases while Conditional Entropy will be used to find the minimum subset of requirements. The test cases and common requirements in TCR table will be reduced without affecting the fault detection capability of the system. The performances of the proposed algorithm and the existing algorithm were compared in terms of reduction rate while preserving the fault detection capability. This research does not cover the processing time for reduction and the complexity for the algorithm.

Data set used in this research is from the Test Log of Automatic Teller Machine System Independent Verification and Validation Project (ATMS IV & V) by Malaysian Software Testing Board (*MSTB Lecturer Aid (Student)*) as in Appendix A. ATMS IV & V project scope of testing is only limited to black-box functional testing for features developed in the test object. This test only addresses system level test which excludes unit test, static test, integration test, regression test and confirmation test.

1.4 Significance of Study

At the end of this research, the proposed technique produced a minimal subset of test cases and requirements. Minimize the subset of test cases and requirements without affecting the decision of the testing is very important to get the quality test cases. Ultimately it is beneficial to optimize time and effort spent on testing and it is also helpful during regression testing (Rothermel *et al.*, 2002). The method to select minimum test case and requirement can be used to design a good and beneficial tool to reduce the size of test cases in every changes of the software. Consequently, this research is conducted to prove that rough set theory and conditional entropy is suitable for one-step-ahead test cases reduction method.

1.5 Thesis Organization

The remaining part of this thesis is broken up into the following chapters. Chapter 2 discusses the relevant background information regarding previous approaches used in test cases reduction in the following order: (1) overview of software testing concept, (2) test case redundancy and its disadvantages in software testing, (3) brief idea on Rough Set Theory and (4) Conditional Entropy used in software testing and (5) several techniques that have been employed in test case reduction .This chapter also elaborates the virtues and limitations of these methods. Chapter 3 describes the methodology used to come out with the proposed test cases reduction method. This chapter provides a clear picture on the research framework. The implementation of similarity relation in Rough Set Theory and Conditional Entropy are also discussed in details. The rationale for integrating Rough Set Theory with Conditional Entropy is presented. Then, the process flow is transferred to formalize the algorithm in Chapter 4. Individual sections detail up the process, starting from creating the Test Cases-Requirement (TCR) Matrix table, and evaluating the performance of the proposed method. The rules and algorithm of the approach will also be discussed in this chapter. The last chapter, Chapter 5, concludes the work done and several recommendations are suggested in order to improve the performance of the proposed approach.

CHAPTER 2

LITERATURE REVIEW

This chapter provides the literature review for this study. General overview on software testing is given in Section 2.1. Section 2.2 discusses the fundamental of Rough Set Theory which is Similarity Relation and Section 2.3 discusses on Conditional Entropy. Test Cases Reduction techniques are discusses in Section 2.4 while comparative analysis of existing test case reduction techniques is presented in Section 2.5. Section 2.6 summarizes the whole Chapter 2.

2.1 Software Testing

Over the decades, as the pervasiveness of software development has increased, testing becomes a critical part of SDLC. Software testing occurs continuously during the SDLC and is considered as part of software quality assurance (SQA) processes. It is done to validate and verify that software meets the needed requirement and to discover any error. The IEEE Software Engineering Body of Knowledge (SWEBOK) states that software testing is a dynamic process and requires selected test cases to verify the behavior of the program (Abran *et al.,* 2004). It is important to find and remove existing faults in the program and preserve its overall quality. In software testing, once a set of requirements is established, a set of test cases (test suite) is generated to fulfill the requirements.

Test cases are always known as one of the challenging tasks in software testing (Singh, 2014; Zeng & Tan, 2012). Test cases are important and need to be provided earlier so that the entire system requirement can be tested. Test case is a set of test input, condition and expected result running to the software to verify the

quality specification of the system (Myers *et al.*, 2011). Results from the test will determine whether or not the system meets user and system specifications. IEEE (Committee, 1983) standard defines a set of test case as "A set of test inputs, execution and expected results developed for a particular objective, such as to exercise a particular program path or to verify compliance with a specific requirements". For example, a test case answers the question: "What am I going to test?" Test cases are developed to define the things that need to be validated to ensure that the system is working properly and is built with high level of quality. The combination of test cases is called a test suite where it contains detailed instruction or goals for each collection of test case to be used during testing (Pomeranz & Reddy, 1997). A test suite consists of all the test cases that satisfy some system requirements. Test cases can be generated automatically using testing tools or manually created by a tester. There are two fundamental approaches in generating test cases, known as functional and structural testing. Functional testing or also called as static or black-box test is based on the view that any component or system can be tested at requirement or implementation level without looking at the internal code structure and knowledge of internal path of the software. In contrast, structural testing or white-box test involves the execution of the software. There are various techniques used to generate test cases for example finite state machine (Soo-In *et al.*, 2000), neural network (Zhao & Lv, 2007), genetic algorithm (Rajappa *et al.*, 2008), soft computing (Mitra & Hayashi, 2000) and many others (Hooda & Chhillar, 2014). There are some techniques that generate test cases based on system requirements. This system produced test data based on requirement of the program. They are also techniques that generate test cases which test the system using some input and the expected output will be derived from condition like UML diagram, critical path, code based test generation technique, GUI based test generation technique, dynamic path testing and evolutionary technique, graph traversal algorithm and genetic algorithm.

Any software contains a set of requirements that should be fulfilled. A test suite consists of all test cases that satisfy all system requirements. Table 2.1 (Mohapatra & Prasad, 2015) is an example that shows test cases in a test suite that satisfies system requirements. Consider a program *P* written to meet a set of requirements *R*; *P* and *R* are denoted as *(P,R)*. Let *T* be a test set containing t_i test cases to test *P* to determine whether it meets all requirements in *R*. The test cases

REFERENCES

Abran, A., Moore, J. W., Bourque, P., Dupuis, R., & Tripp, L. (2004). Software Engineering body of knowledge. *IEEE Computer* Society, Angela *Burgess*, pp. 8.

Alian, M., Suleiman, D., & Shaout, A. (2016). Test Case Reduction Techniques-Survey. *International Journal of Advanced Computer Science & Applications, 1 (7)*, pp. 264-275.

Anwar, Z., & Ahsan, A. (2013). Multi-objective regression test suite optimization with fuzzy logic. *Proceedings of the Multi Topic Conference (INMIC), 2013 16th* International. IEEE. pp. 95-100.

Barr, E., Harman, M., McMinn, P., Shahbaz, M., & Yoo, S.-I. (2015). The oracle problem in software testing: A survey. pp. 507-525.

Binkley, D. (1997). Semantics guided regression test cost reduction. *IEEE Transactions on Software Engineering, 23 (8)*, pp. 498-516.

Black, J., Melachrinoudis, E., & Kaeli, D. (2004). Bi-criteria models for all-uses test suite reduction. *Proceedings of the Proceedings of the 26th International Conference on Software Engineering*. IEEE Computer Society. pp. 106-115.

Campos, J., & Abreu, R. (2013). Leveraging a Constraint Solver for Minimizing Test Suites. *Proceedings of the 2013 13th International Conference on Quality Software*. pp. 253-259.

Chen, S., Chen, Z., Zhao, Z., Xu, B., & Feng, Y. (2011). Using semi-supervised clustering to improve regression test selection techniques. *Proceedings of the 2011 Fourth IEEE International Conference on Software Testing, Verification and Validation*. IEEE. pp. 1-10.

Chen, T. Y., & Lau, M. F. (1998). A new heuristic for test suite reduction. *Information and Software Technology, 40 (5)*, pp. 347-354.

Chen, X., Li, J., Ma, J., Tang, Q., & Lou, W. (2014). New algorithms for secure outsourcing of modular exponentiations. *IEEE Transactions on Parallel and Distributed Systems, 25 (9)*, pp. 2386-2396.

Chen, X., Li, J., Weng, J., Ma, J., & Lou, W. (2016). Verifiable computation over large database with incremental updates. *IEEE transactions on Computers, 65 (10)*, pp. 3184-3195.

Chvatal, V. (1979). A greedy heuristic for the set-covering problem. *Mathematics of operations research, 4 (3)*, pp. 233-235.

DeMilli, R., & Offutt, A. J. (1991). Constraint-based automatic test data generation. *IEEE Transactions on Software Engineering, 17 (9)*, pp. 900-910.

Deris, M. M., Abdullah, Z., Mamat, R., & Yuan, Y. (2015). A new limited tolerance relation for attribute selection in incomplete information systems. *Proceedings of the Fuzzy Systems and Knowledge Discovery (FSKD), 2015 12th International Conference on*. IEEE. pp. 964-970.

Fraser, G., & Wotawa, F. (2007). Redundancy based test-suite reduction. *Proceedings of the International Conference on Fundamental Approaches to Software Engineering*. Springer. pp. 291-305.

Galeebathullah, B., & Indumathi, C. (2010). A novel approach for controlling a size of a test suite with simple technique. *Int. J. Comput. Sci. Eng, 2* pp. 614-618.

Garey MR, J. D. (1979). *Computers and Intractability: A guide to the theory of NP-Completeness*. New York, NY: W. H. Freeman and Company.

Gotlieb, A., & Marijan, D. (2014). FLOWER: optimal test suite reduction as a network maximum flow. *Proceedings of the Proceedings of the 2014 International Symposium on Software Testing and Analysis*. ACM. pp. 171-180.

Gupta, A., Mishra, N., & Kushwaha, D. S. (2014). Rule based test case reduction technique using decision table. *Proceedings of the Advance Computing Conference (IACC), 2014 IEEE International*. IEEE. pp. 1398-1405.

Gupta, R., & Soffa, M. L. (1993). Employing static information in the generation of test cases. *Software Testing, Verification and Reliability, 3 (1)*, pp. 29-48.

Haider, A. A., Nadeem, A., & Rafiq, S. (2013). On the Fly Test Suite Optimization with FuzzyOptimizer. *Proceedings of the Frontiers of Information Technology (FIT), 2013 11th International Conference on*. IEEE. pp. 101-106.

Harris, P., & Raju, N. (2015). Towards test suite reduction using maximal frequent data mining concept. *International Journal of Computer Applications in Technology, 52 (1),* pp. 48-58.

Harrold, M. J., Gupta, R., & Soffa, M. L. (1993). A methodology for controlling the size of a test suite. *ACM Transactions on Software Engineering and Methodology (TOSEM), 2 (3),* pp. 270-285.

Herawan, T., Deris, M. M., & Abawajy, J. H. (2010). A rough set approach for selecting clustering attribute. *Knowledge-Based Systems, 23 (3),* pp. 220-231.

Hirsh, J. B., Mar, R. A., & Peterson, J. B. (2012). Psychological entropy: a framework for understanding uncertainty-related anxiety. *Psychological review, 119 (2),* pp. 304.

Hooda, I., & Chhillar, R. (2014). A Review: Study of Test Case Generation Techniques. *International Journal of Computer Applications, 107 (16),* pp. 105-156.

Huang, C.-Y., Chen, C.-S., & Lai, C.-E. (2016). Evaluation and analysis of incorporating Fuzzy Expert System approach into test suite reduction. *Information and Software Technology, 79* pp. 79-105.

Jeffrey, D., & Gupta, N. (2007). Improving fault detection capability by selectively retaining test cases during test suite reduction. *IEEE Transactions on software Engineering, 33 (2),* pp. 108-123.

Jones, J., & Harrold, M. J. (2003). Test-suite reduction and prioritization for modified condition/decision coverage. *Software Engineering, IEEE Transactions on, 29 (3),* pp. 195-209.

Khalilian, A., & Parsa, S. (2009). Bi-criteria test suite reduction by cluster analysis of execution profiles. *Proceedings of the IFIP Central and East European Conference on Software Engineering Techniques*. Springer. pp. 243-256.

Lin, J.-W., & Huang, C.-Y. (2009). Analysis of test suite reduction with enhanced tie-breaking techniques. *Information and Software Technology, 51 (4),* pp. 679-690.

Mitra, S., & Hayashi, Y. (2000). Neuro-fuzzy rule generation: survey in soft computing framework. *IEEE transactions on neural networks, 11 (3),* pp. 748-768.

Mohammadian, A., & Arasteh, B. (2013). Using Program Slicing Technique to Reduce the Cost of Software Testing. *Journal of Artificial Intelligence in Electrical Engineering, 2 (7)*, pp. 24-33.

Mohapatra, S. K., & Prasad, S. (2015). Finding Representative Test Case for Test Case Reduction in Regression Testing. *International Journal of Intelligent Systems and Applications, 7 (11)*, pp. 60.

MSTB Lecturer Aid (Student). Malaysia Software Testing Board.

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

Nachiyappan, S., Vimaladevi, A., & SelvaLakshmi, C. (2010). An evolutionary algorithm for regression test suite reduction. *Proceedings of the Communication and Computational Intelligence (INCOCCI), 2010 International Conference on*. IEEE. pp. 503-508.

Pawlak, Z. (1982). Rough sets. *International Journal of Computer & Information Sciences, 11 (5)*, pp. 341-356.

Pomeranz, I., & Reddy, S. M. (1997). On the compaction of test sets produced by genetic optimization. *Proceedings of the Test Symposium, 1997.(ATS'97) Proceedings., Sixth Asian*. IEEE. pp. 4-9.

Qin, H., Ma, X., Zain, J. M., Sulaiman, N., & Herawan, T. (2011). A Mean Mutual Information Based Approach for Selecting Clustering Attribute. *Proceedings of the International Conference on Software Engineering and Computer Systems*. Springer. pp. 1-15.

Rajappa, V., Biradar, A., & Panda, S. (2008). Efficient Software Test Case Generation Using Genetic Algorithm Based Graph Theory. *Proceedings of the 2008 First International Conference on Emerging Trends in Engineering and Technology*. pp. 298-303.

Roongruangsuwan, S., & Daengdej, J. (2010). Test case reduction methods by using CBR. *Proceedings of the International Workshop on Design, Evaluation and Refinement of Intelligent Systems (DERIS2010)*. pp. 75.

Rothermel, G., Harrold, M. J., Ostrin, J., & Hong, C. (1998). An empirical study of the effects of minimization on the fault detection capabilities of test suites. *Proceedings of the Software Maintenance, 1998. Proceedings., International Conference on*. pp. 34-43.

Rothermel, G., Harrold, M. J., Von Ronne, J., & Hong, C. (2002). Empirical studies of test-suite reduction. *Software Testing, Verification and Reliability, 12 (4)*, pp. 219-249.

Saif-ur-Rehman, K., Nadeem, A., & Awais, A. (2006). TestFilter: A Statement-Coverage Based Test Case Reduction Technique. *Proceedings of the Multitopic Conference, 2006. INMIC '06. IEEE*. pp. 275-280.

Sampath, S., Bryce, R., & Memon, A. M. (2013). A uniform representation of hybrid criteria for regression testing. *IEEE Transactions on Software Engineering, 39 (10)*, pp. 1326-1344.

Sampath, S., & Bryce, R. C. (2012). Improving the effectiveness of test suite reduction for user-session-based testing of web applications. *Information and Software Technology, 54 (7)*, pp. 724-738.

Shen, Q., Jiang, Y., & Lou, J. (2017). A new test suite reduction method for wearable embedded software. *Computers & Electrical Engineering, 61* pp. 116-125.

Singh, N. P., Mishra, R., & Yadav, R. R. (2011). Analytical review of test redundancy detection techniques. *Int J Comput Appl*, pp. 0975-8887.

Singh, R. (2014). Test Case Generation for Object-Oriented Systems: A Review. *Proceedings of the Communication Systems and Network Technologies (CSNT), 2014 Fourth International Conference on*. pp. 981-989.

Software Engineering Technical Committee (1983). *IEEE Standard for Software Test Documentation*. Institute of Electrical and Electronic Engineers Computer Society (ANSI/IEEE Standard 829-1983).

Soo-In, L., Yongbum, P., Myungchul, K., Hee Yong, Y., & Ben, L. (2000). Automatic test case generation using multi-protocol test method. *Proceedings of the Computer Communications and Networks, 2000. Proceedings. Ninth International Conference on*. pp. 360-366.

Suri, B., Mangal, I., & Srivastava, V. (2011). Regression test suite reduction using an hybrid technique based on BCO and genetic algorithm. *Special Issue of International Journal of Computer Science & Informatics (IJCSI), ISSN (PRINT),* pp. 2231-5292.

Tallam, S., & Gupta, N. (2005). A concept analysis inspired greedy algorithm for test suite minimization. *Proceedings of the ACM SIGSOFT Software Engineering Notes*. ACM. pp. 35-42.

Wang, G. (2002). Extension of rough set under incomplete information systems. *Proceedings of the Fuzzy Systems, 2002. FUZZ-IEEE'02. Proceedings of the 2002 IEEE International Conference on*. IEEE. pp. 1098-1103.

Wong, W. E., Horgan, J. R., London, S., & Mathur, A. P. (1995). Effect of test set minimization on fault detection effectiveness. *Proceedings of the Proceedings of the 17th international conference on Software engineering*. ACM. pp. 41-50.

Xu, S., Miao, H., & Gao, H. (2012). Test suite reduction using weighted set covering techniques. *Proceedings of the Software Engineering, Artificial Intelligence, Networking and Parallel & Distributed Computing (SNPD), 2012 13th ACIS International Conference on*. IEEE. pp. 307-312.

Yan, T., & Han, C. Z. (2014). A Novel Approach Based on Rough Conditional Entropy for Attribute Reduction. *Proceedings of the Applied Mechanics and Materials*. Trans Tech Publ. pp. 1607-1619.

Yoo, S., & Harman, M. (2010). Using hybrid algorithm for pareto efficient multi-objective test suite minimisation. *Journal of Systems and Software, 83 (4)*, pp. 689-701.

Yoo, S., & Harman, M. (2012). Regression testing minimization, selection and prioritization: a survey. *Software Testing, Verification and Reliability, 22 (2)*, pp. 67-120.

You, L., & Lu, Y. (2012). A genetic algorithm for the time-aware regression testing reduction problem. *Proceedings of the Natural Computation (ICNC), 2012 Eighth International Conference on*. IEEE. pp. 596-599.

Zeng, B., & Tan, L. (2012). Test criteria for model-checking-assisted test case generation: a computational study. *Proceedings of the Information Reuse and Integration (IRI), 2012 IEEE 13th International Conference on*. IEEE. pp. 600-607.

Zhao, R., & Lv, S. (2007). Neural-Network Based Test Cases Generation Using Genetic Algorithm. *Proceedings of the Dependable Computing, 2007. PRDC 2007. 13th Pacific Rim International Symposium on*. pp. 97-100.