A T-Way Test Suite Generation Strategy for Sequence Based Interaction Testing

by

Mohd Zamri Bin Zahir Ahmad
1430211379

A thesis submitted in fulfillment of the requirements
for the degree of Master of Science (Computer Engineering)

SCHOOL OF COMPUTER AND COMMUNICATION ENGINEERING
UNIVERSTI MALAYSIA PERLIS

2016

UNIVERSITI MALAYSIA PERLIS

DECLARATION OF THESIS

Author's full name : MOHD ZAMRI BIN ZAHIR AHMAD

Date of birth : 25 MAY 1988

Title : A T-WAY TEST SUITE GENERATION STRATEGY FOR SEQUENCE BASED INTERACTION TESTING

Academic Session : 2015 / 2016

I hereby declarethat the thesis becomes the property of Universiti Malaysia Perlis (UniMAP) and to be placed at the library of UniMAP. This thesis is classified as :

☐	**CONFIDENTIAL**	(Contains confidential information under the Official Secret Act 1972)*
☐	**RESTRICTED**	(Contains restricted information as specified by the organization where research was done)*
☑	**OPEN ACCESS**	I agree that my thesis is to be made immediately available as hard copy or on-line open access (full text)

I, the author, give permission to the UniMAP to reproduce this thesis in whole or in part for the purpose of research or academic exchange only (except during a period of_____years, if so requested above).

Certified by:

SIGNATURE

880525-26-5203
(NEW IC NO. / PASSPORT NO.)

Date : 18 AUGUST 2016

SIGNATURE OF SUPERVISOR

DR. ROZMIE RAZIF BIN OTHMAN
NAME OF SUPERVISOR

Date : 18 AUGUST 2016

ACKNOWLEDGEMENT

Praised be to Allah S.W.T the Most Gracious and the Most Merciful. Peace is upon him, Muhammad messenger of Allah. Alhamdulillah, first and foremost, I would like to express my gratitude to Allah S.W.T for His blessed and mercy, I managed to complete this research work successfully. I also want to thank to all who had guided me through all technical difficulties throughout the project.

Firstly, I'm grateful to have a very supportive family especially my parents, Zahir Ahmad and Ishrat Jahan, and my wife, Rosliza Rasli. Their courage are endless and I will never forget their continuous support, encouragement and praise towards me to complete this course. This research also definitely cannot be successful if I don't have a very knowledgeable and supporting supervisor. He has provided me a nonstop guidance, continuous advice, support and assistance through completion of this project. To my supervisor Dr. Rozmie Razif Othman, I owe you one. I really appreciate your knowledge and support and it means more than you think. Only Allah knows.

Not to forget, my very special thanks is for School of Computer and Communication (SCCE) of Universiti Malaysia Perlis (UniMAP) and Ministry of Higher Education Malaysia (MOHE) for supporting and fund my study under SLAB program. The scholarship is mean for me. Last but not least, for those who lend me their hand within the period of completing this thesis, Mohd Shaiful Aziz, Mohd Wafi Nasrudin and others, may your life is full of brightness and happiness in the future. Thank You.

TABLE OF CONTENT

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

AETG	Automatic Efficient Test Generator
BA	Bee Algorithm
CA	Covering Array
CPU	Central Proccessing Unit
CR	Cooling Rate
EDIST-SA	Event Driven Input Sequence T-way using Stimulated
GB	Giga Byte
GTWay	Generalized T-Way Test Suite Generator
GUI	Graphical User Interface
IPOG	In-Parameter-Order-General
ISTQB	International Software Testing Qualitfication Board
ITTDG	Integrated T-way Test Data Generation Strategy
JDK	Java Development Kit
NA	Not Available
NASA	National Aeronautics and Space Administration
NIST	National Institute of Standard and Technology
NP-Hard	Non-deterministic Polynomial-time Hardness
RAM	Random Access Memory
SA	Stimulated Annealing
SCA	Sequence Covering Array
SCATS	Sequence Covering Array Test Suite Data Generation
SSA	Standard Stimulated Annealing
SUT	System Under Test

TCG	Test Case Generator
TCGA	Test Case Generator Algorithm
T-Seq	T-Sequence
U	Upper Bound
Ur	Upper Bound Reversal

Strategi Penjanaan Sut Ujian T-Hala untuk Ujian Interaksi Berdasarkan Turutan

ABSTRAK

Bagi melengkapi teknik rekabentuk ujian yang sedia ada (contoh; analisis sempadan nilai, pembahagian setara dan graf sebab dan kesan), ujian t-hala adalah suatu teknik rekabentuk ujian yang digunakan khusus untuk mengesan pepijat yg terhasil melalui interaksi input. Sejak 20 tahun yang lalu, banyak strategi t-cara yang telah dicadangkan dalam penyelidikan termasuk General T-way (GTWay), In Parameter Order General (IPOG), Automatic Efficient Test Generator (AETG), dan Jenny. Walaupun strategi t-hala yang telah dicadangkan terbukti boleh mengesan pepijat (dalam banyak penyelidikan kajian kes menunjukkan bahawa keberkesanan sut t-hala ujian setanding dengan sut ujian menyeluruh), strategi ini hanya memberi fokus kepada interaksi input tidak berurutan. Strategi t-hala tidak berurutan adalah mustahil untuk digunakan untuk kawalan dan sistem reaktif (iaitu isyarat input tiba pada masa yang berbeza). Oleh itu, penyelidik pada masa kini mula memberi fokus kepada strategi t-hala berdasarkan urutan. Walaubagaimanapun, penjanaan sut t-hala ujian adalah masalah tidak-berketentuan masa-polinomial (NP-Hard), maka tiada satu strategi boleh menyatakan ia boleh menghasilkan sut ujian yang optimum bagi setiap konfigurasi sistem. Bermotivasikan daripada cabaran tersebut, tesis ini mencadangkan suatu strategi baharu t-hala, yang dinamakan Sequence Covering Array Test Suite Data Generation (SCATS), yang menyokong ujian t-hala berdasarkan urutan. SCATS mengimplementasikan tiga komponen utama iaitu aturan merangkumi turutan, penjana tuple, dan penjana kes ujian untuk menghasilkan saiz sut ujian yang optimum. Penilaian telah dibuat dengan membandingkan SCATS dan strategi yang telah sedia ada dari segi saiz sut ujian, dengan pelbagai kekuatan ($3 \leq t \leq 5$) dan acara ($3 \leq s \leq 30$). Hasil daripada eksperimen ini menunjukkan bahawa dalam purata, SCATS menghasilkan keputusan sut ujian yang berdaya saing berbanding dengan strategi lain.

A T-Way Test Suite Generation Strategy for Sequence Based Interaction Testing

ABSTRACT

Complementing existing test design techniques (e.g. boundary value analysis, equivalent partitioning and cause and effect graphing), t-way testing is a test design technique that specifically used to cater bugs due to interaction. Many t-way strategies have been proposed in literature including General T-way (GTWay), In Parameter Order General (IPOG), Automatic Efficient Test Generator (AETG), and Jenny for the past 20 years. Although proposed t-way strategies have been proven to detect bugs (in many published case studies demonstrate that the effectiveness of t-way test suite comparable to exhaustive test suite), these strategies only focus on sequence-less interaction. For control and reactive system (i.e. input signals arrived at different time), the implementation of sequence-less t-way strategy is not possible. As a result, researchers nowadays start to focus on sequence based t-way strategy. However, as generating t-way test suite is an NP-Hard problem, no single strategy can claims it producing the optimal test suite for every system configuration. Motivated by the aforementioned challenges, this thesis presented a new t-way strategy, named Sequence Covering Array Test Suite Data Generation (SCATS), which support sequence based t-way test suite generation. SCATS implements three main components which is sequence tree, tuple generator and test case generator in order to produe the optimum test suite size. Evaluations have been done by comparing SCATS with existing strategies with various strength($3 \leq t \leq 5$) and events ($3 \leq s \leq 30$) in term of test suite size generated. Experimental result demonstrates that in most cases SCATS produces competitive test suite size compare to other competing strategies.

CHAPTER 1

INTRODUCTION

1.1 Overview Background of Software Testing

Nowadays, many applications use in our daily life (i.e. including both hardware and software applications) are controlled by software. These applications are varied from small, simple, and non-critical application to a very large, complex and life-critical application. In order to fulfill the requirements and needs for the application, the software engineers need to develop a software system that is flexible, tangible and even configurable.

In doing so, the developed software becomes more complicated and complex. This is because current software development needs to get involved with numbers of input parameters, several system integrations and multiple environment deployments. However, to do exhaustive testing is next to impossible due to resources and timing constraints (Younis, Zamli, & Isa, 2008). Here, software testing plays an important role in ensuring that all functional as well as non-functional requirements of the developed software system are fulfilled. Even though software testing usually consume much of resources (i.e. both financial and work hours), reducing efforts in this stage can lead to a greater resource consumptions.

Software testing aims is to determine whether it matches its specifications while executes in its planned environments (Saini & Rai, 2013). In a software development life cycle, the software testing phase is most important to be conducted. Software testing consumes about 40% to 80% of the total cost in the development of software (Manika & Sona, 2014). According to the reports by Charles T. Carroll, lack of testing has cost around 22.2 to 59.5 billion US Dollars every year in United States (Carroll, 2003a), (Carroll, 2003b). This loss figure excluding the loss from critical software application since it is impossible to determine the cost of losing one's life.

Since software nowadays has been entrusted to operate in safety critical and life threatening applications (Othman, Zamli, & Nugroho, 2012), (Zamli, Klaib, Younis, Isa, & Abdullah, 2011), (Kuhn & Reilly, 2002), advancement in software testing is desirable in order to ensure the conformity of the developed software. As a result, many software testing techniques (or strategies) have been proposed in the literature. International Software Testing Qualifications Board (ISTQB) has listed several important testing strategies which include Equivalent Partitioning, Boundary Value Analysis (BVA), Cause and Effect Graphing and Combinatorial Testing (or also refers as Interaction Testing). Researchers invented various testing strategies in order to reduce number of test data. Hence, variants of testing are aims to identifies variants of errors and faults (Shekhar, 2014).

Equivalent partitioning is first introduced by Glenford Myers in his book Art of Software Testing (Myers, 2004). In equivalence partitioning, input domain can be divided into numbers of equivalence classes. This could be done by categorize test cases which have same parameter or behavior into same class. By assuming the test case in

the same classes is equivalence with another value, all other test cases in the same class also would expect to discover the same fault too. By applying this method, software tester could reduce much of testing time as repeated test case is reduced.

There are at least two classes could be defined which is valid and invalid class. Valid class covers all test case which has been defined by the input parameter of a system while invalid class is covered all possible values that might be useful for testing. There are also external conditions that may be identified and categorized in another class. After test case has been categorized, the process of identifying test cases is beginning with a unique number assigned to each equivalence class. After test cases have been covers of all valid equivalence classes, new test cases are produced to cover as many as possible of uncovered valid equivalence classes if any. By summary test case design of equivalence partitioning is depends on two major steps which identifying the equivalence classes and defines the test cases.

For example a company has set up discount for sale, 5% discount is eligible for purchase within RM50 to RM100 and 10% discount is eligible for purchase more than RM100. By using equivalence partitioning, we can classify the test input parameter into invalid and valid classes. For invalid class, the first class is for purchase below than RM50 (e.g. RM49.90). For valid class, the first class is for purchase of RM50 to RM100 and second class is for purchase of RM100 and above. Noticed that there are 3 classes identified and this could ease into the software testing process.

Boundary Value Analysis has been defined by Boriz Beizer (Beizer, 1990). Boundary value analysis also called as range checking, is more likely same concept with

equivalence partitioning except process of creating test cases at the edge of classes. In fact, in a research done by Reid, boundary value analysis is found as most effective compared to equivalent partitioning (Reid, 1997). One of the famous quotes is "Bugs lurk in corners and congregate at boundaries" (Beizer, 1990). It means that fault often happens at the edge of equivalence of classes which is software testers frequently put less attention on this matter. Myers has defined the two major differences of boundary value analysis and equivalence (Myers, 2004), firstly boundary value analysis focuses on one or more test cases at the edge of equivalence classes rather than selecting any test cases in equivalence partitioning as main subject of the test.

Secondly, equivalence partitioning technique it attempt to put a lot of attention on the input condition to derive test cases, but in boundary value analysis test cases also derived from the output equivalence classes' condition. However, equivalence partitioning used together with boundary value analysis in software testing as it is related to each other. There are advantages of using this technique which is software tester will have a very good procedure in determining test cases to be selected. Other than that, boundary value analysis also may expose potential user input problems in order to find the test cases. Test cases generated also will be small in number as all possible test cases at the edge of classes have been derived.

For the same example as given in equivalence partitioning above, there are 3 classes defined which purchase below than RM50, purchase between RM50 to RM100 and purchase more than RM100. For first class the boundary is for RM49.99 and below. The second class is RM50.01 and RM100, and the third class is RM100.01 and above.

In boundary value analysis the final test cases should reflect and covers test cases of this condition at least once.

The Software testing technique Cause-Effect Graph was made-up by Bill Elmendorf of IBM in 1973 (Elmendorf, 1973). Different from equivalence partitioning and boundary value analysis, instead of designing a test case manually, test designer could list the function of inputs into Boolean graph. This can be done by listing all possible relationships between specific combinations of input and output. A combination of inputs is called as cause and the output of the combination is called as effect. This method is also known as fish-bones diagram. After listing, the second step is trying to connect the cause and effect listed with logical annotation such as AND, OR, NOT and so on. There are maybe known cause leads to unknown effect and vice versa, but at this point a dummy node is placed to complete the graph. A table of decision then is constructed based on the completed graph and hence test cases are generated based on it. By doing the cause-effect graphing, two consideration must be made up which is range values and cause constraints. An advantage of using cause-effect graphing is it helps to find the root cause of problems, hence it could reduce the number of test cases. But in real-world applications, the time takes to do such modeling is limited since it consumes longer time to design the analysis for a larger system.

Interaction testing is a form of functional testing (M. B. Cohen, 2004). It is also familiar known as t-way strategies. As today's most of technology is software depended, software testing strategies also need to be concerned. Most of the software consists of components and as the result, faults may occur between unexpected interactions of the inputs. Unlike equivalence partitioning and boundary value analysis,

interaction testing focus on finding bugs due to interactions between input parameter (Zamli, Othman, & Zabil, 2011). Usually this type of faults is seen after the software has been delivered. Many types of t-way strategies have been produced which includes combinatorial testing, pairwise testing, sequence covering array testing and more. In t-way strategies, it requires every combination of any t parameter values to be covered by at least one test, where *t* is referred to as the strength of coverage and usually takes a small value (Lei, Kacker, Kuhn, Okun, & Lawrence, 2007). As the exhaustive test is impossible for a larger system, by using t-way strategies, it could reduce the number of test cases by selecting the suitable strength. The smaller strength picked, will cause the smaller test cases produces.

Many papers published in the past 25 years have shared the practice of utilizing of combinatorial testing or t-way strategies and its effectiveness by testing in real world applications. As examples, ADA compiler was successfully tested by Mandl in 1985 using 2-way testing (Mandl, 1985), (Austin, Wilkins, & Wichmann, 1991). Combinatorial testing also has been used to improve the quality and efficiency of internet protocol testing (Borroughs, Jain, & Erickson, 1994). In year 1996, Williams and Probert demonstrated that combinatorial testing can be used to test the network interfaces (Williams & Probert, 1996). Kevin Burr used combinatorial testing technique to generate test cases to test the email system, with AETG (Burr & Young, 1998). In year 2000, Huller shares the experience of generating a minimum subset of test cases which gave good coverage of the test domain by using the combinatorial testing for system level testing of small, commercial satellite ground systems (Huller, 2000). Moreover, many researchers have reported to use the same technique as Mandl in testing Graphical User Interface (GUI) coding (White & Almezen, 2000), (Xie &

Memon, 2008), (Huang, Cohen, & Memon, 2010), (Yuan, Cohen, & Memon, 2011), (Memon & Xie, 2005). Not only that, researchers from the National Institute of Standard and Technology (NIST) have succeeded in testing a NASA based system using 6-way testing (Kuhn, Lei, & Kacker, 2008), (Kuhn & Okun, 2006).

As mentioned earlier, even a small contribution towards software testing is desirable in order to complement the advancement in current software technology. This research work focus on test case design strategy that based on interaction testing. Further Section in this Chapter will highlight the problem statements as well as the roadmap of the thesis.

1.2 Problem Statements

One of the challenges in software testing is to design a good test case. As a result, many test case design strategies have been proposed in the literature, including boundary value analysis, equivalence partitioning and cause and effect graphing (Myers, 2004), (Hass, 2008), (Sharma & B., 2010), (Naik & Tripathy, 2008). However, these strategies are not focusing for interaction testing problems. While these strategies have proven its usefulness, none of these strategies are suitable to cater bugs due to interaction between input parameters.

Therefore, researchers start to focus on interaction testing, which also known as t-way testing (where *t* refers to interaction strength). As a result, many t-way test suite (consists of several test cases) generation strategies have been proposed recently including Automatic Efficient Test Generator (AETG) (D. M. Cohen, Dalal, Parelius, &

Patton, 1996), (M. B. Cohen, Dwyer, & Shi, 2007), Jenny (Jenkins, 2010), General T-Way (GTWay) (Zamli, Klaib, et al., 2011) and In Parameter Order General (IPOG) (Lei et al., 2007). Despites the above mentioned proposed t-way strategies, these t-way strategies only focus on sequence-less interaction between the input parameters. In control and reactive system (i.e. input signals arrived at different times), usage of sequence-less t-way strategy is not possible.

Not until recently, researchers start to investigate on sequence based t-way strategy. Several sequence based t-way strategies have been proposed in literature (e.g. T-SEQ (Kuhn, Higdon, Lawrence, Kacker, & Lei, 2012) , U and Ur (Chee, Colbourn, & Horsley, 2013), BA (Mohd Hazli, Zamli, & Othman, 2012), and EDIST-SA (Rahman, Othman, Ahmad, & Rahman, 2014)). However, generating a t-way test suite has been classified as a Non-deterministic Polynomial-time Hardness (NP-Hard) problem (Shiba, Tsuchiya, & Kikuno, 2004), (Nie & Leung, 2011) & (Chooramani & Garhwal, 2013).

Motivated by the aforementioned challenges, this research presented a sequence based t-way strategy, named Sequence Covering Array Test Suite Data Generation (SCATS). SCATS strategy is designed to focus for sequence based input interaction testing where its function is to produce a minimum test suite size compared to other existing strategies.

1.3 Aim and Objectives

The aim of this research is to design and evaluate a t-way test suite generation strategy, named Sequence Covering Array Test Suite Data Generation (SCATS). This strategy is specifically designed for sequence based input interaction testing. To comprehend this aim, the following objectives are adopted;

i. To propose a suitable data structure that can be used in generating sequence based t-way test suite.
ii. To design a suitable implementation algorithm in order to generate sequence based t-way test suite.
iii. To evaluate the proposed strategy's preformance (i.e. in terms of test suite size generated) by comparing with benchmark experiment.

1.4 Research Scope

International Software Testing Qualifications Board (ISTQB) highlights 5 main stages in common software testing activities. There are test planning and control stage, test analysis and design stage, test implementation and execution stage, evaluating exit criteria stage and finally control and monitoring stage. All these stages are illustrated in Figure 1.1 below.

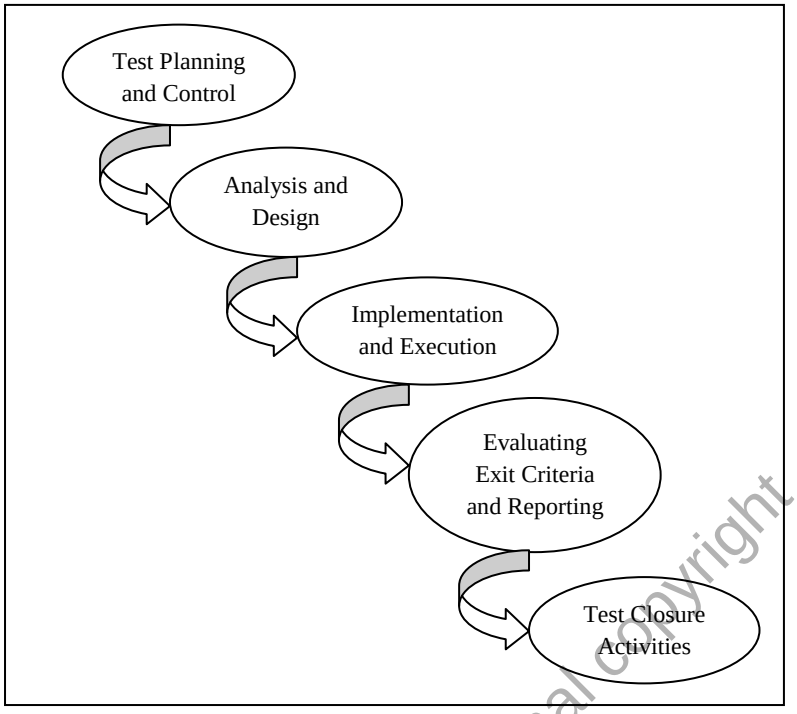

Figure 1.1: Software Testing Cycle.

In test planning, there are three major tasks involve within this stage. First is to determine scope, risks and identify objective of testing. Second is to determine test approach and the third are to implement the test strategy. While in control segment, it involved with several tasks which is assessed and analyzes the result of testing, monitor the test coverage, provides information on testing before to initiate corrective actions, while last step is to make decisions.

Second cycle is test analysis and test design. Principally in this stage it starts with reviewing the test basis in order to identify the test conditions. Then a set of test is designed. After that, the design of the test is evaluated in term of the testability of the systems and requirement.

Third cycle is test implementation and execution. In test implementation it has several major tasks which start with prioritize test cases and creates test data. Then a set of test procedure is listed down for instruction during the test execution. Next step is to create a test suite which contains a set of test case. This is for ensuring the efficiency of the test execution. While in test execution, it has the following tasks which is to execute the test suites based on the test procedures prepared earlier. A retest is executed if there is test that failed and it is part of procedure to determine the fix. Every time test being executed, a log of outcome each of the test is recorded. This includes version of the software under tests. It is called a test log and is used for audit trail. After the test has been done, the result of the test is compared to the expected result.

The fourth cycle is to evaluate exit criteria and reporting. Exit criteria will be set at each test level based on risk assessment and the criteria is vary from project to project. Exit criteria are required when there is maximum test are executed with certain pass percentage. Also used when the bug rate is noticeably below a certain level or when the deadline is achievable. Additionally, there are several tasks involved in this cycle, which is to check and compare test logs and exit criteria specified in the test plan. Then to asses if the exit criteria need to be amend or more tests are needed. The last and most important within this cycle is to write a test summary report for the stakeholders.

The last cycle is test closure activities. This stage is done when software is fully delivered or released to the end customers. There are also other reasons that can affect factor to close the testing such as when the project is cancelled, the target is achieved, maintenance updated done and many more. The following step is a major task involved within this stage, which is to check and confirm the deliverable product is based on

actual demand. Then testware such test script must be ensured to be finalize and archived because it may be will be used later. The most important in this cycle is to handover the testware to the maintenance support organization in order to give full support of the software in future.

In this research, all works focus test on test analysis and design stage. This research aims to modify and enhance an existing technique of test suite generation in order to support the sequenced based input parameter interaction. In addition, the strategy needs to be applied must have the abilities to create a minimum test cases as possible in the test suite without compromising with t-way testing principle which is to cover all input parameter interactions for at least once. Furthermore, in this research scope, applications that being tested with this strategy is in range of event, *s* between 3 to 30 while the strength, *t* varies from 3 to 6. Then the test suite generated by the proposed strategy will be executed in Test Execution stage.

1.5 Research Methodology

In this research, there are three main phases of research methodology which can be classified as;

i. Literature Review
ii. Design and Development
iii. Evaluations