

Measuring Test Case Reusability
Based on Simplicity and
Independence

By

Mr. Mohammad Rava

Submitted in Partial Fulfillment of the
Requirement for the Degree of
Master of Science
in Information Technology
Assumption University

July, 2013

Measuring Test Case Reusability Based on Simplicity and Independence

By

Mr. Mohammad Rava



**Submitted in Partial Fulfillment of the
Requirement for the Degree of
Master of Science
in Information Technology
Assumption University**

July, 2013


The Faculty of Science and Technology

Master Thesis Approval


Thesis Title A Model for Measuring Test Case Reusability
By Mr. Mohammad Rava
Thesis Advisor Asst. Prof. Dr. Jirapun Daengdej
Academic Year 1/2013


The Department of Information Technology, Faculty of Science and Technology of Assumption University has approved this final report of the **twelve** credits course. **IT7000 Master Thesis**, submitted in partial fulfillment of the requirements for the degree of Master of Science in Information Technology.

Approval Committee:

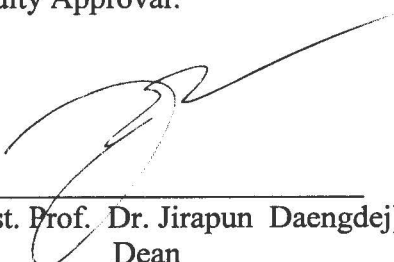

(Asst. Prof. Dr. Jirapun Daengdej)
Advisor


(Asst. Prof. Dr. Vichit Avatchanakorn)
Committee Member


(Asst. Prof. Dr. Thotsapon Sortrakul)
Committee Member


(Assoc. Prof. Dr. Surapong Euwatanamongkol)
Commission of Higher Education
Ministry of Education

Faculty Approval:


(Asst. Prof. Dr. Jirapun Daengdej)
Dean

July / 2013

ACKNOWLEDGEMENTS

First and foremost, praises and thanks to God the Merciful and Compassionate to whom I owe my very existence. Without His blessings of wisdom and perseverance, none of this would have been possible.

I would like to express my deep and sincere gratitude to my Adviser, Asst. Prof. Dr. Jirapun Daengdej for giving me the opportunity to do this research and providing me with invaluable guidance throughout the journey. His dedication and sincerity to his work and students has deeply inspired me to become the best in my field of study. With only a few words he was able to convey me volumes of knowledge that have helped carrying out this thesis. It was truly an honor to work and study under his guidance. I am beyond grateful for all that he has offered me. I also would like to extend my thanks to his wife and family for their patience and understanding for all the time he had given to me and my research at hours that were beyond his responsibility to me.

I am extremely grateful to my parents for their love and prayers. Without their encouragement and support none of this would have been possible. Particularly my mother, Dr. Fatemeh Hamidifar for sharing her experiences and knowledge on research and teaching me about the methodology that helped me a great deal in my research.

I would also like to thank the Assumption University, the Faculty of Science and Technology in particular, for providing the tools that made all this possible.

Finally, I would like to thank all the people who made it possible for me to complete the research, whether directly or indirectly.

ABSTRACT

Software Testing is a very important part of Software Engineering; however its effectiveness is reflected on the amount of time and resources that it is spent on. To reduce the consumption of already scarce resources, software engineers have come up with a solution known as reuse. One of the most common components for reusability is the test case. However there are very few that concentrate on making a metrics standard model for test cases let alone its reusability, and the few that exists are either complex or flawed in nature, particularly when measuring all forms of test cases.

Based on previous studies done in the field of test case reusability this research focuses on creating a model for measuring test case reusability. A general template is provided by other researchers. However each research focuses on a single aspect of reusability, and does not concentrate the main criteria relevant in measuring the reusability of the test case. Hence this study proposes a model for measuring test case reusability.

TABLE OF CONTENTS

The Faculty of Science and Technology Thesis Approval	i
Acknowledgements.....	ii
Abstract.....	iii
Table of Contents.....	iv
List of Figures.....	vii
List of Tables	viii
1. Chapter 1: Introduction.....	1
A. Overview.....	1
B. Goals and Objectives	3
C. Scope and Limitations.....	4
2. Chapter 2: Literature Review.....	5
A. Introduction.....	5
1. Test Case.....	5
2. Use Case	10
3. Reusing Test Cases.....	14
B. Related Work	16
1. A Study of Reusability and Complexity.....	17
2. Test Case Reusability Metrics Model.....	19
a) Reusability Factors.....	19

b) The Drawbacks	29
3. Test Case Reduction Methods by Using CBR.....	31
a) Measuring Test Case Complexity	31
b) Drawbacks.....	34
C. The Statement of Problem	36
1. Issues on Measuring Understandability.....	37
2. Issues on Measuring Changeability.....	40
3. Issues on Measuring Universal.....	41
4. Issues on Measuring Complexity	43
3. Chapter 3: Measuring Test Case Reusability.....	44
A. The Solution.....	44
1. Modified Test Case Reusability Metrics Model.....	44
2. Simplicity and Reverse Complexity.....	44
3. Model for Measuring Test Case Reusability	50
4. Chapter 4: Evaluation	51
A. Method of Evaluation	51
B. Evaluation Samples.....	53
1. Course Registration System.....	54
2. The ATM Withdraw Based on Activity Diagram	64
3. Hotel Management System Guest Login.....	74
5. Chapter 5: Results and Discussion.....	79

A. Course Registration System.....	80
B. ATM Withdraw Based on Activity Diagram.....	82
C. Hotel Management System Guest Login	83
D. Test Case Reusability Metrics Model Comparison	84
6. Chapter 6: Conclusion and Further Work.....	85
A. Conclusion	85
B. Drawbacks.....	87
1. Lack of Metrics Weight.....	87
2. Additional Factors	87
3. Limitations Due to Scope of Work.....	88
C. Further Study	89
7. References and Bibliography	90
8. Appendix A: Programming Codes.....	95

LIST OF FIGURES

Figure 2-1: Use Case Diagram for a University Course Registration System [26] 11

Figure 2-2: Basic and Alternate Flow of Events in a Use Case [26] 13

Figure 2-3: Control Flow Diagram Example [51] 31

Figure 2-4: Abnormal Control Flow Example [54] 34

Figure 3-1: Control Flow Example [51] 45

Figure 4-1: Login Process Test Flow [17] 51

Figure 4-2: Course Registration Use Cases [26] 54

Figure 4-3: Course Register Control Flow 57

Figure 4-4: ATM Withdraw Activity Diagram [55] 64

Figure 4-5: ATM Withdraw Activity Diagram Graph and Control Flow [55] 66

Figure 4-6: Generated Test Paths for ATM Withdraw [55] 67

Figure 4-7: HMS GUEST LOGIN Control Flow 74

Figure 4-8: HMS Guest Login Test Paths 75

LIST OF TABLES

Table 2-1: HM Invalid Guest Login Test Case.....	8
Table 2-2: Textual Use Case Template.....	12
Table 2-3: Thresholds of Human Cognition	21
Table 2-4: Parameters Calculation Formulas.....	22
Table 2-5: Invalid Test Case	39
Table 3-1: Test Case Flow Simplicity	48
Table 4-1: “Course Register” Test Case Matrix	59
Table 4-2: Register Course Test Links and Test Items.....	60
Table 4-3: Course Register Test Case Independence	61
Table 4-4: Simplicity Measurement Results.....	62
Table 4-5: Course Register Test Case Reusability	63
Table 4-6: ATM Withdraw Test Case Values	67
Table 4-7: ATM Withdraw Test Cases 1, 2 and 3	68
Table 4-8: ATM Withdraw Test Cases 4 and 5	69
Table 4-9: ATM Withdraw Test Cases 6 and 7	70
Table 4-10: Independence Measurement Results	71
Table 4-11: Simplicity Measurement Results.....	72
Table 4-12: Reusability Measurement Results	73
Table 4-13: Test Case Required Data Table	75
Table 4-14: Sample Test Cases from Hotel management System Login Process	76

Table 4-15: Independence Measurement Results77

Table 4-16: Simplicity Measurement Results.....77

Table 4-17: Reusability Measurement Results78

Table 5-1: Linkert Scale for Test Case Reusability79

Table 5-2: Course Register Test Case Reusability80

Table 5-3: ATM Withdraw Reusability Measurement Results82

Table 5-4: HMS_GL Test Case Reusability Results83



1. CHAPTER 1: INTRODUCTION

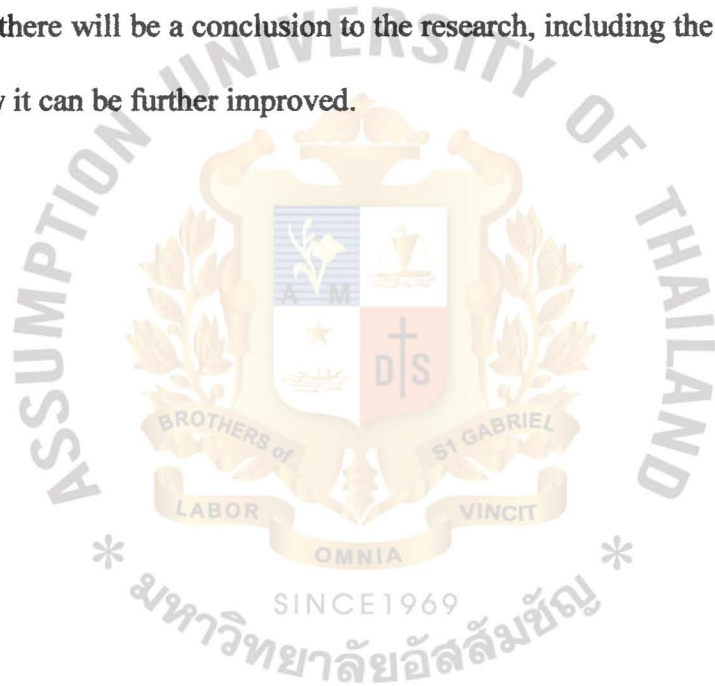
A. Overview

A common practice to reduce the cost of test development is through reusability [19], particularly test case reusability. This practice helps testers to avoid duplicating their efforts in order to create the same test case, doing so will also improve the quality of software testing and greatly reduce the cost of production which would lead to further enhancing the productivity of software companies [15].

In recent years many research institutions and software companies have studied test cases, each have studied a specific aspect of the test case. In Domain based testing: Increasing test case reuse [20], the researchers attempted on creating a test case generation method based on the concepts of software reuse, domain analysis and domain modeling. A similar ideology can be seen in “Domain based testing: A reuse oriented test method” [22]. In “Test Reuse in CBSE Using Built in Test” [21], they present an architecture which is derived from Component Based Software Engineering (CBSE) in which they integrate built-in tests in software components that in turn makes it possible to reuse tests. These researchers have concentrated on the management and generation of test cases, and they often have dealt with test suits as a whole. Measurement of test case as a singular component has seldom been studied, let alone the measurement of test case reuse.

This research concentrates on measuring the test case reusability, and makes it quantifiable on how reusable a test case it is as a sole, and not as a group or a suit. The research presented is based from many previous researches and articles; however the main aspect of the research has come from “A Study of Reusability, Complexity and Reuse Design Principles” [40], “Test Case Reusability Metrics Model” [10], and “Test Case Reduction Methods by Using

CRM” [51]. In Chapter 2 a basic background is given of the main aspects of test cases and reusability, and also each of the main literature are explained in detail and how they are relate to this research. In Chapter 3 issues regarding each of the criteria are disclosed and how the issue affects the system is shown with examples and proof of concept. A solution to the issues is also provided the second part of Chapter 3, including how all the new criteria come together to create the new reusability measurement model. In Chapter 4 methods of evaluating the model is disclosed alongside three separate samples from three different sources that generate test cases. In Chapter 5 the results are evaluated and discussed, and finally in Chapter 6 there will be a conclusion to the research, including the drawbacks of the new system and how it can be further improved.



B. Goals and Objectives

This research concerns with creating a model for measuring test case reusability. By analyzing several researches, this study finds the different issues with each existing model and provides an alternative solution. The main goal of this research is:

1. To discover several potential criteria for measuring reusability in software systems, and identify potential reusability factors for test cases in software testing.
2. To create a basic expandable template for measuring test case reusability based on independence and simplicity for test cases generated from control flow diagrams and use cases.
3. To evaluate the template by using different samples for generating test cases based on control flow diagrams and use cases.

C. Scope and Limitations

This research has the following scope and limitations:

1. This research focuses on measuring test case reusability for test cases that are generated from use cases and control flow diagrams.
2. The test cases have to be designed with white box testing as well as black box testing in mind, due to the requirements it has for measuring an aspect of reusability, there needs to be a certain degree of knowledge on the procedure of test case generation such as use cases.
3. The current template for measuring test case reusability includes only two factors as main measurement attributes and for further precision it requires more criteria to be added.
4. The test case template does not have an innate weight system among the factors, meaning that in its current stage all factors are considered equal in importance when measuring overall reusability.

2. CHAPTER 2: LITERATURE REVIEW

A. Introduction

Software engineering is comprised of several processes with each falling into a certain phase which eventually builds the Software Development Life Cycle (SDLC). There are many different kinds of SDLC. However all of them follow the same logic as seen in Progressive SDLC [25]. It starts with Planning, Analysis and Design, moving to Development, Implementation and finally Maintenance or testing. Different methods often split these processes into smaller more specified ones, or some even integrate it together. In this research we concentrate on Software Testing, which is mainly located in the Maintenance phase, however in most recent SDLCs it is believed the this components should exist across all other phases and should run parallel to them. This further emphasizes the importance of Software Testing.

IEEE defines software testing as a process that analyzes a software item to detect the differences of existing and required conditions and evaluate the features of the software item [1]. Bugs, errors and defects are the result of the detection process. Software testing has been estimated to take as much as 70% of the overall cost of producing the application or software [2]. This implies the importance of software testing in software development process.

1. Test Case

An essential component of software testing is test case. The Institute of Electrical and Electronics Engineers defines a test case as "A set of test inputs, execution conditions, and expected results developed for a particular objective, such as to exercise a particular program path or to verify compliance with a specific requirement." [5].

A test case is essentially a mechanism in which a tester will use to evaluate if a software or application is functioning properly. The mechanism is usually a set of conditions or variables under which the tester will use to make those evaluations.

The purpose of a test case is to find defects [6]. In very general terms the very aim of a test is to run and trigger failures that expose defects. In developer language, these defects are regarded and often called “bugs”. Test cases are involved in many types of testing; usually they are grouped into test suites and are used in testing either a whole software system or a particular component.

According to IEEE 829-1998 [29] a test case is comprised of many components and specifications which are used to completely address the workings of the testing components.

The first component is “Test Case Specification Identifier” which is a unique generated number or name related to the software components in testing. The Test Case Identifier – most commonly referred to as Test Case ID is usually a short unique name for the case which could include any relevant numbering such as versioning number, date, sequence and etc.

The second component is “Test Items” which identifies the items that are to be tested in the test case. Such items are usually included in the requirement specification, system design specification as well as other relevant guides, manuals and documentation.

The third component is “Input Specification”, which are all inputs required to execute the test case. The inputs usually consist of data, values, variables and even files.

The fourth component is “Output Specification”, which are considered to be all the outputted necessary to verify the test case. The outputs can be data elements and values, but also it can

be human actions, conditions, files and relationships, depending on the level of test case the specification may vary.

The fifth component is “Environmental Needs”, which are mainly the hardware and software requirements necessary for the test case to properly function. It also can include necessary facilities and training.

The sixth component is “Inter-case Dependencies”, which identifies any prerequisite test cases. The precursor should often identify all prerequisites.

These components aid in creating the basic model of a test case. In this research we mainly concentrate on certain values that will most likely aid in measuring test cases, by using these components and components seen in other relevant researches such as the Test Case Reusability Metrics Model [10] we use the following attributes:

1. Test Case ID: Based on “Test Case Specification Identifier” [29], it is used to create a unique identifier for each test case which in this research consists of three main parts shown in the following format: “Abbreviated name for the system/application _ system/application sub-identifier _ Sequence Number”
2. Test Item: The item that is being tested from the system/application. Based on the second component “Test Items” [29].
3. Test Case Objective: The purpose of the test case, sometimes referred to as Test Case Description, used to describe the test case in a few simple words.
4. Test Case Keyword: Keywords used for the purpose of searching test cases, this particular attribute is based on Test Case Reusability Metrics Model [10] which is used as a metrics component.

5. **Test Inter-Case Dependency:** Based on the sixth component, and required in the measurement of test case reusability according to TCRMM [10]. This component mainly keeps track of the number of precursor test cases.
6. **Test Steps:** Steps to execute in order to successfully perform a test case.
7. **Test Date:** Based on “Input Specification” [29], which mainly tracks the values involved in the verification of the test case.
8. **Expected Result:** Based on “Output Specification” [29], which is used to verify the result of the test case, mainly used as comparison with actual result. A test case only passes when the actual results and expected result are matching.

Table 2-1: HM Invalid Guest Login Test Case

Test Case ID:	HM_GL_01
Test Item:	Username Box, Password Box, Display Message
Test Case Objective:	Invalid Login - Blank Username and Password.
Test Case Keyword:	Login, Verification, Invalid, Username, Password, Message Box
Test Precursors:	0
Test Data:	Username [Invalid] Password [Invalid]
Test Sequence: (Steps to Execute)	<ol style="list-style-type: none"> 1. Enter Login Page 2. Input Username. 3. Input Password. 4. Click on Login 5. Confirm Error Message 6. Input Username
Test Expected Result:	Error Message: Invalid Credentials. Return to Login Page.

A sample of a test case used in this research can be seen in Table 2-1 HM Invalid Guest Login Test Case. The test case is based on Hotel Management System [23] Login Page designed for Guests. The test case ID is HM_GL_01, which stands for Hotel Management System, Guest Login Sub-system and sequence 1 in the test case bundle related to sub-system Guest Login. Test Item in this test case is the Username Box, Password Box and Display Message, mainly since the inputting of the username and password are of main phases of the test and also the result of the test involves in showing a message box. Test Case Objective is Invalid Login resulting from incorrect input of credentials. Test Case Keywords are particular keywords used for the purpose of searching and identification. Test Precursors are the number of test cases precursor to the current test case, meaning mainly the number of test cases that needs to be tested before the current test case is to be tested properly. Test Sequence, also known as Steps to Execute or simply Test Steps are steps and sequences of actions that need to be taken in order for the test case to be successfully executed. Test Expected Result is the expected output of the test case, if the given test case is to be executed successfully.

In order to generate a test case, the user needs access to the use case of a system which will grant him different sequences and pathways to be tested in a specific sub-system or function. By using use cases there will be no function without a test case.

2. Use Case

In a software development project, a use case defines software requirements [26]. The use case describes the developing system's behavior under various situations and conditions as the system responds to a request from a user or a stakeholder which are known as primary actors [27]. To accomplish a certain goal in the system, the primary actor begins an interaction and the system then responds accordingly based on the interest of the user. Depending on a particular request and the conditions surrounding that request, different sequences of behavior or scenarios can unfold. The collection of those different scenarios is a test case. The creation of use cases begins early in the system development. According to IBM Rational Unified Process (RUP) "a use case fully describes a sequence of actions performed by a system to provide an observable result of value to a person or another system using the product under development." [28]

In general Terms a use case tells a customer what to expect, a developer what to code and more importantly it tells the tester what to test. In case of software testing which consists of several related tasks each with their own set of deliverables, test case creation is the first fundamental step. Then test steps are designed for these test cases and finally test scripts are created to implements those steps [26]. Test cases aid in identifying and communicating the conditions that will be implemented in the test and are essential for verifying successful and acceptable implementation of the product requirement, thus resulting in test cases being the key to the whole process.

Fundamentally use cases are written in text form, although they can also be written using flow charts, sequence charts, diagrams and programming languages. In basic level they are used as a communication from one person to another, often with no special training, hence why at early stages of development "simple text" is considered the preferred choice [27].

80096 e.1

Use case diagrams are based on Unified Modeling Language (UML) and are used to represent use cases visually [26]. To explain the workings of use case diagrams we use an example created by IBM Rational's requirements management evangelist [26] and will take it as partial basis for some of the results seen in later chapters.

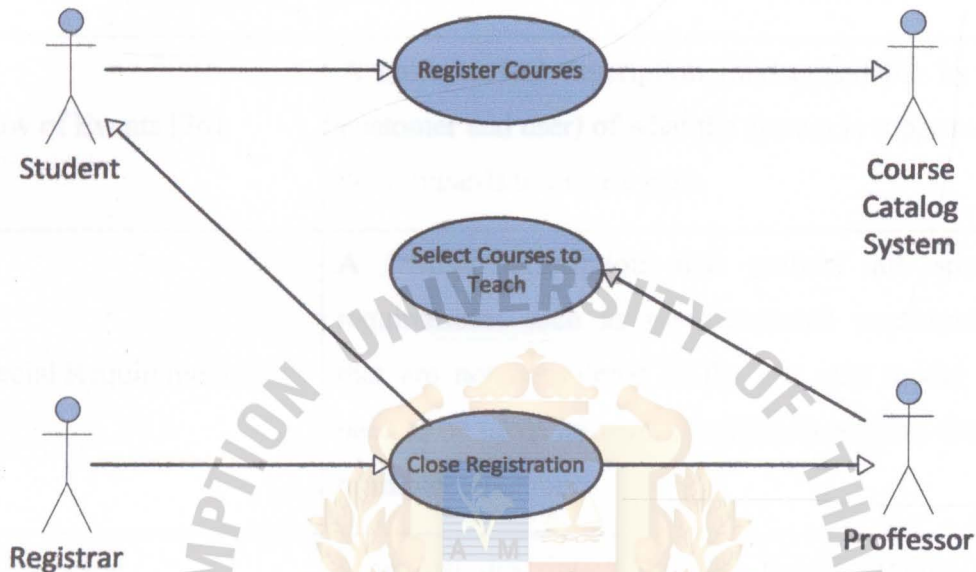


Figure 2-1: Use Case Diagram for a University Course Registration System [26]

In Figure 2-1 a use case diagram depicts requirements for a university course registration system. The stick figures represent “actors” which can be a variety of things, such as people or even other systems that interact with the main system. The ovals represent “use cases”; each use case is a piece of functionality that is to be implemented. The lines connecting the actors and use cases represent communication between the two entities [26, 27].

Each use case requires to be described with a significant amount of text. The text should follow a specific format which usually describes how the use case operates. Table 2-1 is a how an average textual use case is often depicted [27].

Table 2-2: Textual Use Case Template

Use Case Section	Description
Name [26]	The name associated with the use case.
Brief Description [26]	The description of purpose and role of the use case.
Flow of Events [26]	A basic textual description (understandable by the customer and user) of what the system is supposed to do in regards to the use case.
Special Requirements [26]	A textual description that gathers all special requirements such as non-functional requirements that are not considered in the use case model but need to be taken into consideration during the design or implementation.
Preconditions [26]	A textual description defining the constraints and preconditions for the system at the time of use case inception.
Post conditions [26]	A textual description that defines any constraints on the system or conditions that happen at the time the use case ends.

In order to generate test cases from use cases, flow of events in a use case must be created.

There are two components in the flow of events; the first is “Basic Flow of Events”, which covers what normally should happen when a certain use case is performed. The second is “Alternate Flow of Events” which covers behavior of an optional character relative to normal behavior and also variations to that behavior.

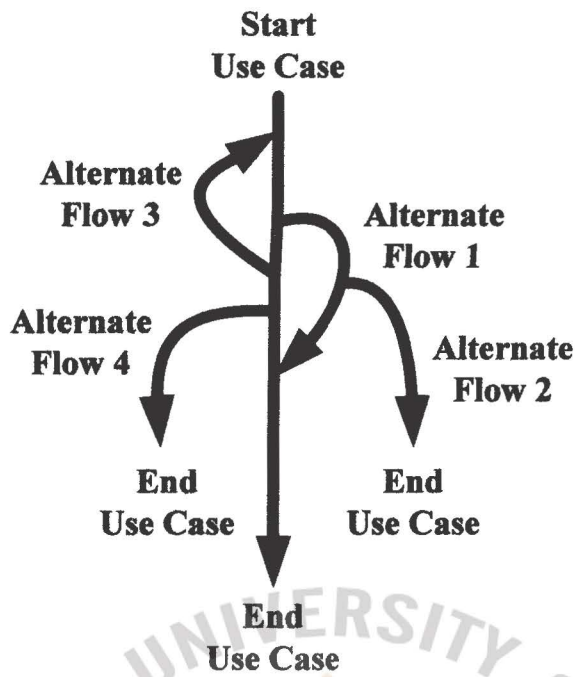


Figure 2-2: Basic and Alternate Flow of Events in a Use Case [26]

Figure 2-2 represents a typical structure for flow of events. The straight arrow is the basic flow, and the curved arrows represent the alternate flow. Some alternate flows such as alternate flow 1 and 3 return the basic flow of events at some point while others such as alternate flow 2 and 4 simply end the use case.

According to IBM Rational Guide [26] there are three steps to follow in order to generate test cases from a use case. The first step is to generate the scenarios by identifying each combination of basic and main flow. Then in the second step we identify the test cases from scenarios. By analyzing each scenario in both textual and diagram form there should be at least one test case for each scenario. The third and final step identifies the values that need to be tested in the test case, since without test data test cases cannot be executed or implemented in any form. After values have been identified a test case is then generated based on the given data. In this research this is the main method used for extracting test cases as it is the most common method recognized for test case generation.

3. Reusing Test Cases

In software engineering, reusability refers to using modules, classes, functionalities and even segments of the code again with little or no modification. The main purpose is to reduce implementation time and decrease the chance of bugs and errors appearing, since prior testing on those modules has refined them. Many studies and researches from the computer science and software industry have analyzed the benefits of software reuse and reusability and believe that it plays a key strategic factor in improving software quality, productivity and reliability as well as reduce development cost [30, 31, 32, 33, 34, 35, 36 and 38]. Doug Mcilroy [37] in 1969 presented component based development for software reuse, suggesting that software components which are interchangeable pieces should form the basis for software systems.

Software reuse has become very popular due to wide application and implementation of object oriented methods and component based development. However recently reuse research on software testing has started to grow compared to the earlier stages. Design and creation of effective test cases is considered an important aspect in software and system testing [39].

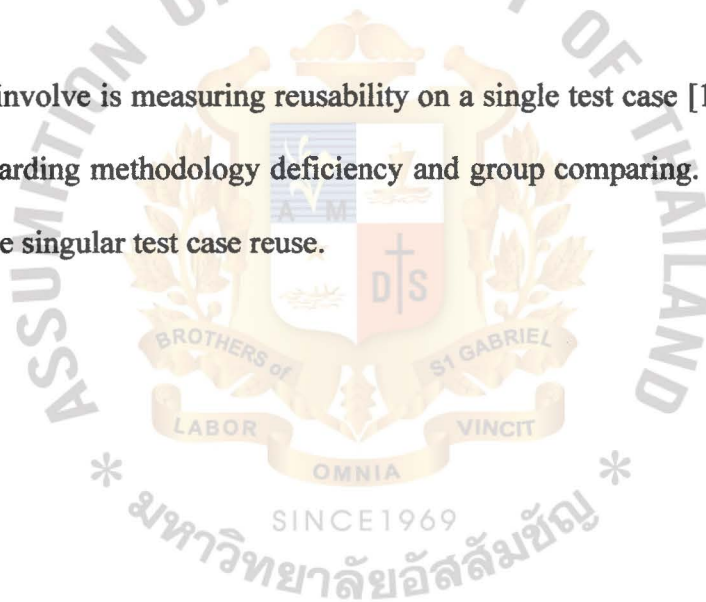
For the purpose of reuse, test cases are often stored in a library of test cases in order to be used as a resource in future applications or systems. A key issue in test case reuse is the effective test case organization and management, which particularly include the depiction and analysis of the test case [40].

Designers should test the system as early and as often as possible and should incorporate different aspects of testing throughout the design process. Even in design process, component reuse does significantly reduce design time and product cost [40]. This is made possible by

reusing components with built-in test information, which ultimately results in a manufactured and developed system with improved quality, and enhanced reliability and maintainability.

The test methodology used will greatly affect the design process. However the test methodology of the system depends strongly on the tests used for each component of that system [40]. In essence the methodology defines what and effective test case is, if methodology is effective then effective test case will be extracted, however if the methodology is defective or even partially defective then as rule of thumb the selected test cases for reuse will also have a chance of being defected. Thus reusing based on methodology often has its cons and downsides.

Other forms of reuse involve is measuring reusability on a single test case [10]. This method removes the issue regarding methodology deficiency and group comparing. In this research we mainly focus on the singular test case reuse.



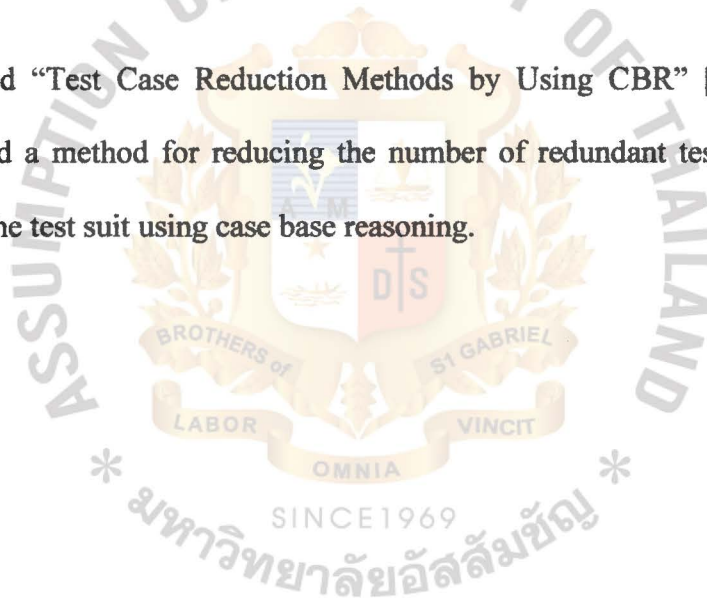
B. Related Work

There are three studies that are used as main references for this research. Their techniques and theories are the basis of the reusability measurement model.

The first study, titled “A Study of Reusability, Complexity and Reuse Design Principles” [36] in which analyzes the components relative to reusability and explores three reuse principles that are imperative for any model that intends on measuring reusability.

The second study titled “Test Case Reusability Metrics Model” [10] in which the researchers devised a metrics model based on their experience to measure the reusability of test cases.

The third study titled “Test Case Reduction Methods by Using CBR” [51] wherein the researchers developed a method for reducing the number of redundant test cases and thus reducing the size of the test suit using case base reasoning.



1. A Study of Reusability and Complexity

The researchers based their study on analyzing reusability of code components in an application. By using a 5 point likert scale very similar to a previous research [41]: (1 – Not Used, 2 – Difficult to Reuse, 3 – Neither Difficult not easy to reuse, 4 – Easy to Reuse, 5 – Very Reusable.) they measured reusability of a component as perceived by subjects reusing the components.

In a previous study done by the same researchers [42] selected subjects built one-use steaming components [43]. Based on a set of reuse design principles the subjects were trailed on software reuse design and converted their one-use components to be reusable. They concluded that the reusable component were significantly larger in size compared to the one-use (single use) components. Following up the study they identified three commonly used reuse design principles which were: “Well-identified Interface”, “Documentation” and “Clarity and Understandability” [36, 42].

Well-Defined Interface: According to the study, an interface is what determines how a components can be reused and how it is interconnected with other components. If the component's interface is simple, it should be easier to reuse. These interfaces have three types which are: Application Programming Interface, User Interface and Data Interface [44].

Documentation: An essential part of any system is documentation which is used for any future use or modification for maintainability. The documentation should be extensible (meaning that it should have the capacity to be expanded upon), adaptable (can be modified depending on the situation) and self-contained (complete and independent) [44].

Clarity and Understandability: Basically in regards to functionality, a degree to which a component is easily understood is referred to as Clarity and Understandability of the

component. Another research [45] discusses the definiteness of a characteristic to make components reusable. Definitiveness is defined the degree of clarity to which the module's purpose, capability, constraints, interfaces, and required resources are defined.

The study follows the common belief that the larger the component the harder is it for it to be reused. According to widely used cost estimation model known as COCOMO II [46], it is considered that software reuse and reusing components cost is higher if the reusable component is larger.

In the study, thirty-four subjects participated, with each subject reusing only 5 components which resulted in a total of 170 reused cases. The assignment of components was randomly selected from a pool of 25 components which were designed and build specifically for reuse.

Using regression analysis the relationship between the complexity of a component and the ease of reuse was analyzed. An inverse correlation was found between the complexity and ease of reuse, which means the higher the complexity, the lower the ease of reuse. However the relationship was not statistically significant. There was also an analysis on the relationship between the reuse design principles and the ease of reuse. The results indicated that of the three reuse design principles, two significantly increased the ease of reuse while the other did not have a significant impact. The ones that had a greater effect on the ease of reuse were Well-Defined Interface and Clarity and Understandability. This means that documentation had a low impact on ease of reuse.

Although this research is not directly related to test case reuse, it does set strong principles in case of reusability which can set a direct correlation for any form of reusability in software system; this also includes reusability in testing and test cases.

2. Test Case Reusability Metrics Model

As organizations implement systematic reuse test cases in software testing to improve productivity and quality, they must be able to measure reusability of test case and identify the most effective reuse strategies. In the Test Case Reusability Metrics model [10] the researchers developed a metrics model for test case reusability, and provided reusability factors that helped establish reusability assessment model.

Reusability has been defined as the degree to which a thing can be used [13]. Test case reusability refers to the test case can be used in a variety of application level. A metric is a quantitative indicator of an attribute of a thing [13]. It is crucial to determine what reusability factors of test case are, and to quantify these reusability factors.

Currently there are no standards for test case metrics, let alone the reusability factor. The researchers referenced ISO/IEC 9126 Software engineering-Product quality [14] and their experience in the field of software testing as their guide to create the Test Case Metrics.

a) Reusability Factors

The researchers that devised TCRMM [10] divided reusability into four main factors. They are understandability, changeability, independence and universal.

In order to measure the factors using an example we take test case HM_GL_03 regarding hotel management system login system in Table 2-1. All the variables are extracted from the source Hotel Management System Project [23].

(1) Understandability

It has been theorized in [10] that the more a test case is understood, then the more it is likely to be reused, and vice versa. The understandability factor measures how easy a test case is understood in terms of its internal and external decryptions. There are several aspects that reflect on Understandability, they are "Test Case Summery" (S), "Keyword" (K), "Test Item" (I), and so on.

There are three formulas used to calculate the understandability of the test case, each based on a certain aspect. The formulas follow the same logic in calculating understandability, however the first formula delves in percentage of understand properties on test case summery, the second formula investigates is the percentage of understand properties on test case keyword, and the last formula uses the same reasoning to find the percentage of understand properties on test case items, also interpreted as test steps. The researchers used thresholds of human cognition as variable to measure the understandability of each criterion.

The first formula in regards to understanding is U_s , which is defined as the percentage of understand properties on test case summary of test case "t". The number of characters in test case summary is depicted with "s" [10]:

$$U_s(t) = \begin{cases} 0, & s = 0 \\ \frac{LS_2 - LS_1}{|2s - LS_1 - LS_2|}, & s \in (0, LS_1) \cup (LS_2, +\infty) \\ 1, & s \in [LS_1, LS_2] \end{cases}$$

The second formula is the percentage of understand properties to test case keyword and a test case "t", which is shown with U_k . The number of keywords in a test case is showing with "k":

$$U_K(t) = \begin{cases} 0, & k = 0 \\ \frac{LK_2 - LK_1}{|2k - LK_1 - LK_2|}, & k \in (0, LK_1) \cup (LK_2, +\infty) \\ 1, & k \in [LK_1, LK_2] \end{cases}$$

The third formula - U_I is the percentage of understand properties to test steps of a test case "t". Number of test items is shown with "i" [10]:

$$U_I(t) = \begin{cases} 0, & i = 0 \\ \frac{LI_2 - LI_1}{|2i - LI_1 - LI_2|}, & i \in (0, LI_1) \cup (LI_2, +\infty) \\ 1, & i \in [LI_1, LI_2] \end{cases}$$

The parameters in the above formulas can change depending on their use and application. The thresholds α , β , and γ are valued in Table 2-3 according to the characteristics of human cognition and linguistic features.

Table 2-3: Thresholds of Human Cognition

Thresholds	α_1	α_2	β_1	β_2	γ_1	γ_2
Value	0.3	0.4	0.3	0.4	0.3	0.4

The Table 2-4 shows how each parameter is calculated in detail. The values related to the parameters are obtained from arithmetic mean of statistical data in the researcher's test case library that includes hundreds of existing test cases.

Table 2-4: Parameters Calculation Formulas

Parameter	Calculation Formula	Value
LS_1	$\frac{1}{n} \sum_{i=0}^n S_i \times (1 - \alpha_1)$	21.17
LS_2	$\frac{1}{n} \sum_{i=0}^n S_i \times (1 - \alpha_2)$	42.38
LK_1	$\frac{1}{n} \sum_{i=0}^n S_i \times (1 - \beta_1)$	2.25
LK_2	$\frac{1}{n} \sum_{i=0}^n S_i \times (1 - \beta_2)$	4.49
LI_1	$\frac{1}{n} \sum_{i=0}^n S_i \times (1 - \gamma_1)$	6.77
LI_2	$\frac{1}{n} \sum_{i=0}^n S_i \times (1 - \gamma_2)$	13.55

To calculate the Understandability factor, the values of U_s , U_k and U_i are shown in the following formula:

$$U(t) = \frac{U_s + U_k + U_i}{3}$$

In order to calculate understandability, we need to first extract the necessary data from the test case (Table 2-1). There are mainly three values that are of concern for understandability. The first value is Test Case Summery, which is also seen as Test Case Description or Test Case Objective, and it is represented by the letter “S”. The second value is the Test Case Keyword, represented by the letter “K”, and the third value is Test Items, represented by the letter “I” which is also seen as number of items being tested such as test data.

The number of characters in Test Case HM_GL_01 (Table 2-1) Test Case Summery is equal to thirty-seven (37) without considering spaces between the words. The formula indicates:

$$U_s(t) = \begin{cases} 0, & s = 0 \\ \frac{LS_2 - LS_1}{|2s - LS_1 - LS_2|}, & s \in (0, 21.17) \cup (42.38, +\infty) \\ 1, & s \in [21.17, 42.38] \end{cases}$$

By allocating the parameters from Table 2-4 it is seen that “s” is a value between the two margins and thus resulting in the Understandability value for Test Case Summery to be 1 meaning 100% understandable.

$$s = 37 \rightarrow s \in [21.17, 42.38] \rightarrow U_s(t) = 1 \rightarrow 100\%$$

The number of keywords in the Test Case HM_GL_01 (Table 2-1) is six (6). Putting the values in the formula results in:

$$k = 6 \rightarrow k \in (0, 2.25) \cup (4.49, +\infty)$$

$$U_k(t) = \frac{LK_2 - LK_1}{|2k - LK_1 - LK_2|} = \frac{2.24}{5.26} = 0.42 \rightarrow 42\%$$

The number of Test Items in the Test Case HM_GL_01 (Table 2-1) is indicated to three (3).

The username, the password and the display message are the 3 elements involved in the test.

According to the formula:

$$i = 3 \rightarrow k \in (0, 6.77) \cup (13.55, +\infty)$$

Since three (3) is lower than 6.77, then the formula is used to calculate the amount which is:

$$U_l(t) = \frac{LI_2 - LI_1}{|2i - LI_1 - LI_2|} = \frac{13.55 - 6.77}{|6 - 6.77 - 13.32|} = \frac{9.78}{14.32} = 0.682$$

$$U_l(t) = 0.682 \rightarrow 68\%$$

The final understandability measurement is the average of the three values. Using the following formula the overall understandability of the test case is measured.

$$U(t) = \frac{U_s + U_K + U_I}{3} = \frac{1 + 0.42 + 0.682}{3} = 0.7$$

$$U(t) = 0.7 \rightarrow U(t) = 70\%$$

Based on the produced value the understandability of the test case HM_GL_01 is 70%, and is considered to be understandable.



(2) Changeability

Test case changeability is possible if the structure and style of the test case are made in a way that the changes can be implemented easily, completely and consistently. The changeability of a test case is directly tied to its data representation.

The researchers believe that the less the number of constants are and the more variables exist, then the higher is its changeability.

To calculate changeability we consider C_c to be a percentage of changeability property to constant of a test case "t". The number of constants in test case "t" is shown with "c":

$$C_c(t) = \begin{cases} \frac{1}{c}, & c \neq 0 \\ 0, & c = 0 \end{cases}$$

The percentage of changeability property to variable of test case "t" is shown as C_v , and the number of variables test case "t" has is shown with "v":

$$C_v(t) = \frac{v}{1 + v}$$

The values of C_c and C_v calculate the changeability factor in the following way:

$$C(t) = \frac{C_c + C_v}{2}$$

To evaluate Changeability we use the same test case for Understandability. Test Case HM_GL_01 is a manual test case and thus it does not have constants represented in outside of the test case template. The two variables it uses are username and password. The following is the result of the formula and value allocation.

Since there are no constants in the formula, then:

$$C_c(t) = \begin{cases} \frac{1}{c}, & c \neq 0 \\ 0, & c = 0 \end{cases} \rightarrow C_c(t) = 0$$

$$C_c(t) = 0$$

And thus v (the variable will be two):

$$C_v(t) = \frac{v}{1+v} = \frac{2}{3}$$

Changeability is then measured by the average of the two values:

$$C(t) = \frac{C_c + C_v}{2} = \frac{0 + \frac{2}{3}}{2} = \frac{2}{6} = \frac{1}{3} = 0.\overline{33} \rightarrow C(t) = 33.33\%$$

Most manual test cases in form of tables do not have a constant. However constants are introduced mainly in automated Test Cases when they are linked together via a template, and thus using resources from that template.

(3) Independence

The researchers assume that the stronger is a test case's independence from other test cases, the more reusable it is. The independence of a test case is measured based on its dependency on other test cases. The more precursor test cases a test case has, then the less independent it is. Chained test cases are a simple example of precursor test cases, when a test case requires a previous test case to be executed before they are executed, then that test case is dependent on its previous test case, and would reflect on its poor independence factor.

To calculate independence, the variable "I" is considered a percentage of independence property of test case "t", and the number of precursor test cases which test case "t" has is represented with "p":

$$I(t) = 2^{-p}$$

In order to evaluate independence we use the Test Case HM_GL_01 (Table 2-1) as reference. According to the test case, there are no precursor test cases to the test case HM_GL_01. Thus the result is:

$$I(t) = 2^{-p} = 2^0 = 1 \rightarrow I(t) = 1 \rightarrow I(t) = 100\%$$

However if there were more test cases precursor to the aforementioned test case, then the result would change depending on the number of precursors. For example, if there were 2 precursor test cases the indolence would decrease.

$$I(t) = 2^{-p} = 2^{-2} = \frac{1}{2^2} = \frac{1}{4} \rightarrow I(t) = 0.25 \rightarrow I(t) = 25\%$$

(4) Universal

The researchers assume that if a test case is more universal, then its reusability is higher. Universal is defined more in terms of test case environment. The universal factor is reflected from test fields and test scenarios that a test case is executed. A test scenario is the testing environment that includes software and hardware environment. An example would be the necessity of having specific software and hardware in order to proceed with the test. a test field on the other hand, refers to application of the test case.

To calculate Universal factor, consider UN_F as the percentage of universal properties to applications of the test case "t". The number of application in which test case "t" can be used is shown with "f":

$$UN_F(t) = \begin{cases} 0, & f = 0 \\ 1 - \frac{1}{f}, & f \neq 0 \end{cases}$$

In test case "t" the percentage of universal properties to software and hardware are associated with UN_W . The number of software scenarios that test case "t" is run is shown with "s", and for the number of hardware scenarios run on the test case "t" the letter "h" is represented.

$$UN_W(t) = \frac{1}{s + h + 1}$$

Both values of UN_F and UN_W contribute to calculate the value of Universal factor.

$$UN(t) = \frac{UN_F + UN_W}{2}$$

b) The Drawbacks

There are several drawbacks related to the Test Case Reusability Metrics Model which are listed as below:

1. The Test Case Reusability Metrics Model is based on the experience researchers had in the field of reusability and test case evaluation rather than empirical evidence [10]. No other references are used in order to assess any of the factors other than user experience.
2. The factor understandability has flaws in regards to language barrier and standards. Some organizations have different methods of expressing test case summery, some others that use an automated method do not have traditional summery designated and use other methods to describe the test case as seen in WebInject [24]. There is also explanation for different language descriptions. The same test case in Spanish will use more words to describe the same test case, while other language such as Chinese may use lesser characters, thus creates an inconsistency among different languages and understandability of the test case. This is mainly due to the fact that the authors assumed meaning exist within the description and test cases, and thus this factor only works when the degree of meaning is assumed to be of full value.
3. Changeability formula is a paradox and a contradiction to the theory of what the formula should do, and thus it is fully unreliable. There is also no indication in the research on the origin of the formula provided. This is mainly due to the fact that the test case generation technique [10] used by the authors has a constant value at all times. The assumption for no constants was presumed for cases in which constants do not happen, however even in such scenario, the formula is logically flawed.
4. In many cases the universal value is not feasibly calculated, since many of the test cases do not store the information regarding the hardware and software scenarios.

This is particularly problematic if test cases are designated before development when the values in regards to the hardware and software scenarios are none existent.

5. Complexity of the test case is not at all included or mentioned as a factor for the reusability, it is well established in many other researches [36, 51]. Thus the main formula for calculating reusability is incomplete in its essence.

In the next section, we elaborate on these drawbacks and provide a new reusability metrics method for calculating the reusability of test cases.



3. Test Case Reduction Methods by Using CBR

In order to reduce the number of redundant test cases the researchers [51] use path coverage criteria in order to reduce test case redundancy. The importance of the complexity in test cases has been seen in other similar researches [53, 54] in which are used by the researcher.

a) Measuring Test Case Complexity

For the purpose of Test Case Reusability Metrics, our main concentration is the method this research uses in order to calculate complexity of a test case. In order to do so, the researchers use Control Flow Graphs which are derived from the source code or the application. Since the method is based on white box testing, each state is assumed to be a block of code. By using path oriented test case generation techniques the researchers use a template control flow to create several different test cases. Figure 2-3 is a control flow diagram used by the researchers [51] in order to extract various variables, including test cases.

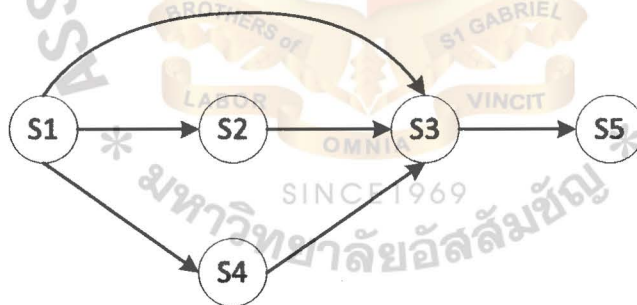


Figure 2-3: Control Flow Diagram Example [51]

Let $S = \{s1, s2, s3, s4, s5\}$ to be a set of stages in the control flow diagram (Figure 2-3). By assuming that each of the above states could potentially reveal a fault, thus it is believed that the ability of five states is to reveal five faults. Since every single transaction must be tested, then in this example each test case will be a traverse of the stages. The result would be:

$$TC_n = \{s_1 \rightarrow s_2 \rightarrow s_3 \rightarrow \dots \rightarrow s_n\}$$

In which TC_n is considered to be a test case, and s_n is considered the state which acts as a node in the path oriented graph that is used for the purpose of testing, which in this scenario would be Figure 2-3. Following the pattern, the Control Flow (Figure 2-3) would generate the following test cases based on different stage transactions:

$$TC_1 = \{s_1 \rightarrow s_2\}$$

$$TC_8 = \{s_1 \rightarrow s_4 \rightarrow s_3 \rightarrow s_5\}$$

$$TC_2 = \{s_1 \rightarrow s_3\}$$

$$TC_9 = \{s_2 \rightarrow s_3\}$$

$$TC_3 = \{s_1 \rightarrow s_4\}$$

$$TC_{10} = \{s_2 \rightarrow s_3 \rightarrow s_5\}$$

$$TC_4 = \{s_1 \rightarrow s_2 \rightarrow s_3\}$$

$$TC_{11} = \{s_3 \rightarrow s_5\}$$

$$TC_5 = \{s_1 \rightarrow s_2 \rightarrow s_5\}$$

$$TC_{12} = \{s_4 \rightarrow s_3\}$$

$$TC_6 = \{s_1 \rightarrow s_4 \rightarrow s_3\}$$

$$TC_{13} = \{s_4 \rightarrow s_3 \rightarrow s_5\}$$

$$TC_7 = \{s_1 \rightarrow s_2 \rightarrow s_3 \rightarrow s_5\}$$

In order to measure complexity, the researchers consider the following:

$C_{plx} = \{High, Medium, Low\}$ where C_{plx} is the complexity of the test case. To measure the value of the complexity into the three values (high, medium and low) the researchers devised the following calculation method.

- High Complexity: When the number of states is higher than the average number of states in the test suit.
- Medium Complexity: When the number of states is equal to the average number of states in the test suit.
- Low Complexity: When the number of states is lower than the average number of states in the test suit.

The average number of stages in the thirteen test cases is as follows where “ ns_i ” is the number of stages in the test case I, and Avg_s is the average of stages:

$$Avg_s = \frac{\sum_{i=1}^{13} ns_i}{13} = \frac{35}{13} = 2.69 \cong 3 \rightarrow Avg_s = 3$$

Following the average of stages, the result of the test case complexity would be:

$$C_{plx}(TC_1) = Low$$

$$C_{plx}(TC_8) = High$$

$$C_{plx}(TC_2) = Low$$

$$C_{plx}(TC_9) = Low$$

$$C_{plx}(TC_3) = Low$$

$$C_{plx}(TC_{10}) = Medium$$

$$C_{plx}(TC_4) = Medium$$

$$C_{plx}(TC_{11}) = Low$$

$$C_{plx}(TC_5) = Medium$$

$$C_{plx}(TC_{12}) = Low$$

$$C_{plx}(TC_6) = Medium$$

$$C_{plx}(TC_{13}) = Medium$$

$$C_{plx}(TC_7) = High$$

The complexity of the test case indicates how difficult each test case is to execute [51]. This method does provide a good approach on how to measure test case complexity. However this method does have some shortcomings and drawbacks.

b) Drawbacks

Although the measurement system is a good way to measure test case complexity, it does however have some shortcomings that prevent us from directly using it for measuring test case reusability.

The measurement formula used for measuring complexity of a single test case is based on average of stages in the control flow. In short, if the control flow is abnormal, then the average cannot be representative of the entire test case group. An example of an abnormal control flow is seen in Figure 2-4.

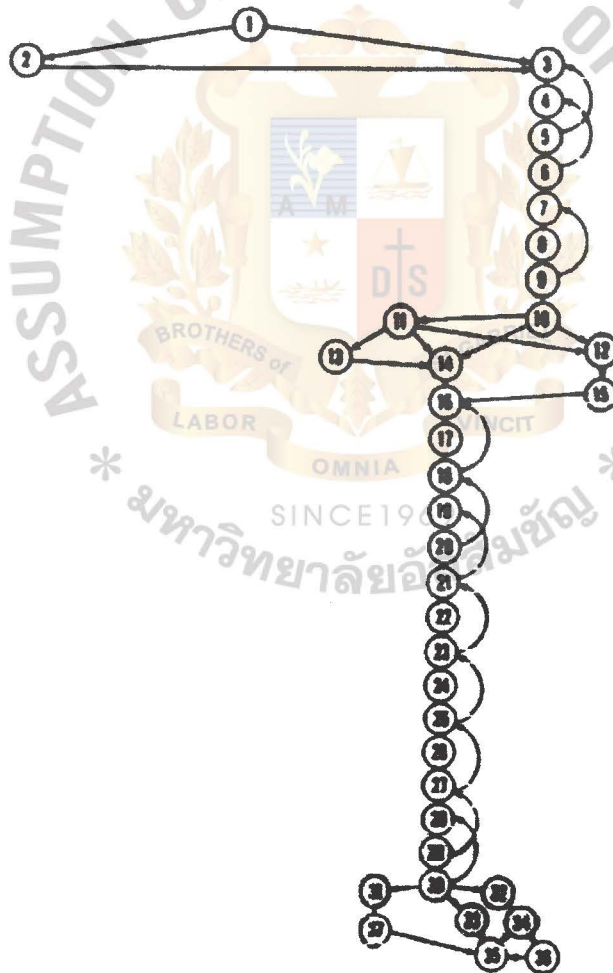


Figure 2-4: Abnormal Control Flow Example [54]

Several test cases can be extracted from Figure 2-4. It can be as low as 2 nodes or even as high as 40 nodes depending on how it is traversed. This imbalance can increase the average significantly and result in some of test cases to measure as low complexity, while they should have been classified as medium or even high. However the concept is relatively sound and it will be expanded upon in chapter 3 where complexity will be measured.



C. The Statement of Problem

Test Case Reusability Metrics was among the very first proposed models that created a quantifiable framework to measure test case reusability [10]. However the metrics factors considered are not without problem.

In order to prove that some of the TCRMM don't work as planned, we used two types of methods to deduct them. First is "comprehensive reasoning", by using references and logical reasoning to deduct some of the factors results, and the second is "Reject by Example", in which we test the TCRMM against some testcases and show why that particular factor fails.

The test case used is from Hotel Management System [23] – which for the purpose of this thesis we are going to focus on the login system which is often considered the most common part of any software system. The codes for the login system are written in java (Appendix A). The standard test case format is applied using the template obtained from Chapter 2.

There are several issues and drawbacks that this research intends on resolving. The first three are issues encountered with factors presented by "Test Case Reusability Metrics Model" [10] which are Understandability, Changeability and Universal by using logical and empirical evaluation to the possible extent. The fourth issue is the missing component from the TCRMM which has been regarded in my other researches as an essential factor for measuring reusability in any form according to the study on reusability and complexity and reuse design principles [40].

1. Issues on Measuring Understandability

The researchers have quantified understandability; but the components used for measurement are incomplete. TCRM Understandability consists of “Keyword Understandability”, “Test Item Understandability”, and “Test Summery Understandability”. However a similar measurement methodology has been used for measuring software understandability, which is a grander scale in terms of size. The research “A Model for Measuring Software Understandability” [15] focuses on measuring several aspects of understandability in regards to software. The purpose of measuring software understandability is to remove the aspect of creating faults that are caused from misunderstandings. For example, an application is written by programmer X. Programmer Y wants to continue developing the application and create a second version. However, Programmer Y does not understand fully what programmer X has written, and hence it will cause programmer Y to misinterpret some factors within the software and create unintentional faults within the second version of the software. To avoid such situation from happening, the researchers created a full list of things that need to be understood before proceeding with the situation.

The first criterion is “Understandability of The Documentation”. An application or software is not maintained only trough source code, but it requires an integrated use of both source code and documentation [16]. In software testing, the test case itself is often regarded as the test documentation, this means that test keyword, and test item and test summery are considered as document side of the code.

The second criterion is "Understandability of Structure" [15]. Without understanding the structure of a test or a software system, it is impossible to reconstruct it correctly. If a software system or a test case is difficult for a software/test engineer to understand, then in would take many times until he/she reconstructs it correctly. TCRMM has not included a

factor for test case structure understandability, and hence it lacks precision when it comes to understanding accuracy measurement in general test cases.

The other criteria are “Understandability of Components”, “Understandability of Data” and “Understandability of Source Code”. However in the scale of a test case it would be detrimental to measure those factors into consideration since the time and resources needed would be rather exponential. It means that in order to measure the full understanding of a single test case, then there must also be understanding of the source code and data spatial complexity which would defeat the purpose of reusing the test case itself.

On another note, considering that only descriptive factors such as test item, test summery and keyword are used to measure understanding, the it would mean that the understanding factor itself would heavily rely on the person at work. Understanding depends not only on understandability of the test case or software system, but also the level of comprehension of the software tester or engineer [15].

The authors of TCRMM [10] used results from their previous paper “Reusable Test Models and Application Based on Z Specification” [7]. The test cases produced in the research were mainly in Chinese language [7], hence makes the measurement less accurate without fully measuring the language understandability.

In order to evaluate the point, we use an invalid test case sample to demonstrate how the arbitrary the metric system is. The test case is in a form of a sample. The words can be replaced with any other words and it would not change the final result. As long as the number of keywords, items and summery characters are within the boundary, then the result would not stay the same.

Table 2-5: Invalid Test Case

Test Case ID:	TC_Invalid
Test Items:	Item1, Item2, Item3, Item4, Item5, Item6
Test Case Objective:	This is a description unrelated to the test case.
Test Case Keyword:	Keyword1, Keyword2, Keyword3
Test Precursors:	0
Test Data:	Data1, Data2, Data3

Following the boundaries presented in Table 2-5 we come to the following conclusions:

$$s = 40 \rightarrow s \in [21.17,42.38] \rightarrow U_s(t) = 1$$

$$k = 3 \rightarrow k \in [2.25,4.49] \rightarrow U_K(t) = 1$$

$$l = 6 \rightarrow s \in [6.77,13.55] \rightarrow U_l(t) = 1$$

$$U(t) = \frac{U_s + U_K + U_l}{3} = \frac{1 + 1 + 1}{3} = 1 \rightarrow U(t) = 100\%$$

Following the formula the result would yield 100% understandability on any given test case as long as it falls into the boundary. Even if the words are randomly generated and replaced with the template words, the test case understandability would still yield a 100% result.

The reason such result is generated is due to the fact that the authors and researchers of test case reusability metrics model assumed a degree of meaning and interpretation in the test case. They did not assume that by any point invalid explanations can be used in order to generate a test case. It is only logical that they assumed that the test case designer would write with the concept of meaning in mind.

Understandability is a valid factor, since according to “A Study of Reusability, Complexity and Reuse Design Principles” [40] the researchers concluded and Understandability and Clarity plays a great role in reusability, however the metrics formula the researchers of TCRMM use has been proven to be inaccurate by example. Hence it’s why Understandability should be removed as a metrics factor from the final reusability formula.

2. Issues on Measuring Changeability

Changeability formula is logically flawed and does not in any way explain how they were obtained. An example on how the formula fails is as following. The researchers believe that the more changeable a test case is, the more reusable it will be [10]. They proceed to assume that the constants which cannot be changed reduce changeability. The more constants exists in a test case, the less the changeable it will be. However the formula contradicts that idea:

$$C_C(t) = \begin{cases} \frac{1}{c}, & c \neq 0 \\ 0, & c = 0 \end{cases}$$

It mentions that if there are no constants, then the changeability of the constants is equal to zero (0) meaning that without constants the changeability is nonexistent, however earlier they mention that the less the constants are the more changeable they are. So the question is that: how is it that no constants would produce the same value as infinite constants?

$$\text{if } c = 0 \text{ then } C_C(t) = 0$$

$$\text{if } c = \infty \text{ then } C_C(t) = \frac{1}{\infty} \cong 0$$

$$\text{if } c = 1 \text{ then } C_C(t) = 1 \rightarrow 100\%$$

This means that 1 constant is more changeable than 0 constants. Even though they specifically point out that constants reduce changeability. This means that formula does have a mathematical error that does not fit the concept and thus it makes the calculation baseless. Also there is no direct indication of how would the formula behave for manual test cases that do not have any constants. If the formula is implemented in such situation, the value for changeability would never reach more than 49.99%.

Although the formula is logically flawed, the test cases that were used by the researchers indicate that the generation technique would always yield a number of constants even though there might be no variables, there will always be constants. However this gesture could have been better demonstrated by simply pointing out that the value of constants can never be equal to zero, rather than the opposite.

3. Issues on Measuring Universal

The researchers have not explicitly defined what they mean by Universal. It has been mentioned that the more universal a test case is, the more reusable it will be. Then they proceed and refer universality to test scenarios and test fields [10]. They measure the number software and hardware scenarios a test case is executed in, but also they measure the application requirements necessary to run the test case.

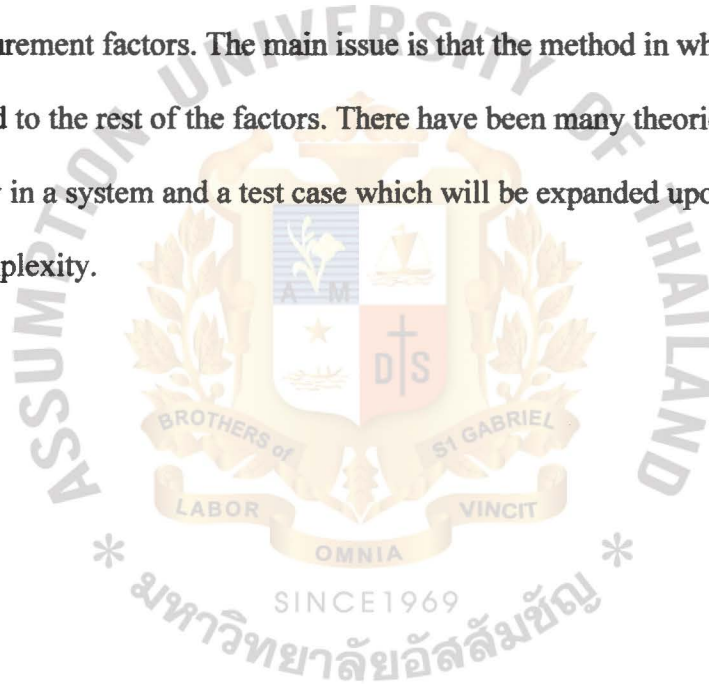
Based on “A New Approach to Generating High Quality Test Cases” [17], in order to create a high quality test case an important aspect that should be viewed is the structure of the component that is being tested. They relate it to the number of inputs and outputs generated from a single process, and how a test case is written to satisfy every aspect of that structure. Hence here we conclude that Universality is too vague.

Another issue with universal is the fact the information required is too detailed. When Rothermel and Harrold developed their “Safe and efficient Algorithm for Regression Testing” they mentioned that the best reusable test case is one that does not require information too expensive to extract [4]. Prior knowledge about the code or development cycle can be very difficult to find specially if the Test Cases are being imported from software developed by a different group of programmers. Technical information such as Hardware and Software scenarios are also difficult to be generated unless they are made part of the Test Case itself, and that would also increase the cost of making test cases. Calculating the Universal Factor effectively would only result in increase of time and resources which makes the whole point of reusing the test cases completely unnecessary.

The authors of TCRMM specifically mention that universal depends on the number of hardware and software requirement needed for a specific test case to run [10]. However this harbors several issues of its own. First, this would mean that there is no possible way to test pre-development test cases (test cases designed before the code development of the system), this is mainly due to the fact that there aren’t any estimated hardware and software scenarios at that phase of the development. The second issue is that the number of hardware and software scenarios, alongside the number of scenarios does not indicate any information about the complexity of the scenarios – if the test case requires a single application that is difficult to apply or a function that is complex to implement, it can be equally reusable as a test case with several easy to install applications and functions. There is also no weighing system for the functions and softwares, as some software and hardware are unique to the situation, then they are more difficult measure and balance in the formula. Thus the number of hardware and software scenarios cannot reflect the simplicity or reuse or how universal and general the test case is.

4. Issues on Measuring Complexity

Complexity is a factor that is not considered in the Test Case Reusability Metrics Model, however many researches on reusability including the study on reusability and complexity [40] which state that the more complex a system is, the harder it is to reuse. Other researchers also determined that the more complex a reuse component is, the higher the cost of reuse will be [52]. In a related research that develops a method on reducing test cases based on case base reasoning [51], it is shown that complexity plays a great role in test case reusability. Thus it is concluded that complexity is a justified factor and should be included in the test case reusability measurement factors. The main issue is that the method in which Complexity is measured and added to the rest of the factors. There have been many theories in regards to measuring complexity in a system and a test case which will be expanded upon in the next section regarding complexity.



3. CHAPTER 3: MEASURING TEST CASE REUSABILITY

A. The Solution

1. Modified Test Case Reusability Metrics Model

Based on the issues with Understandability, we move to exclude it from the test case reusability factors by reason of example and proof. Universal also is not computable based on the provided evidence, since there is no limit to the number of hardware and software scenarios. Also according to the reusability study [40], documentation has proven to be a minimal factor of reusability and have little to no effect on the overall reusability of the components. Changeability is also not mathematically matching with the proposed theory, thus it cannot be accepted as well. The only factor that remains is Independence which is going to be used as one of the prominent factors for measuring test case reusability.

2. Simplicity and Reverse Complexity

There are several techniques proposed for measuring complexity. The study on reusability, complexity and reuse design principles [36] used “Source Lines of Code” (SLOC) as measurement terms. SLOC is among the most popular and most used methods of measuring size and complexity in software metrics. Complexity of Software components has been measured in several empirical studies [47, 48, 49 and 50]. However source lines of code is not feasible in test cases which do not have direct access to the source code, especially in the cases of white box testing, thus the concept of modularity is more relevant to test case metrics.

Modularity [18] has proven that simplicity and clairvoyance in design makes reusability easier, particularly in software reuse. The same logic can be applied to test case; the simpler a

test case is the easier it would be for the test engineers to reuse it. This is mainly because most of the test cases are subject to a small redesign, and if a test case is too complex, then it would take more resources to change it than creating an entirely new test case.

Among the most prominent and reliable methods of measuring complexity in software systems is cyclomatic complexity, which is a software measurement technique developed in 1976 by Thomas J. McCabe [54]. This method is based on software control flows and it mainly measures the complexity of the system by how big and how diverse it is. The more conditions and states exist in a system, the more complex the system will be. Figure 3-1 is an example of a relatively simple control flow from the research on “Test Case Reduction Methods by CBR” [51].

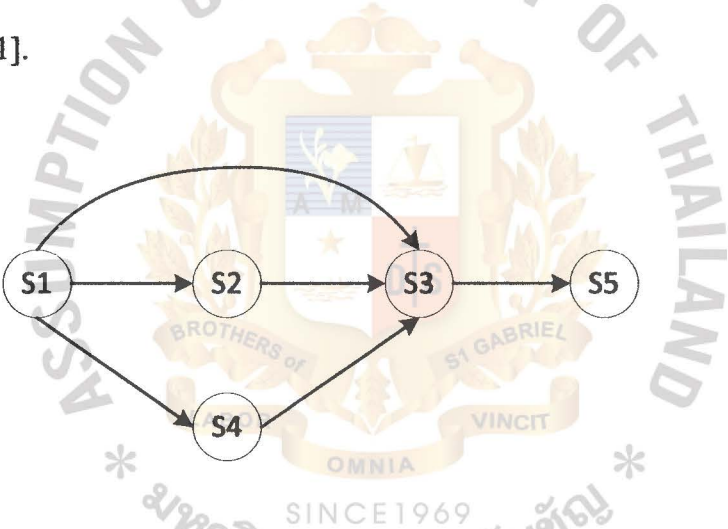


Figure 3-1: Control Flow Example [51]

In order to calculate the system’s cyclomatic complexity, the following formula is introduced by the researcher Thomas J. McCabe [54]. Consider “CC” as the value of Cyclomatic Complexity, then “E” is equal to the number of edges (connection between the nodes) in the control flow, and “N” as the number of nodes. The cyclomatic complexity of the control flow “A” is:

$$CC_A = E - N + 1 = 6 - 5 + 1 = 2 \rightarrow CC_A = 2$$

However in case of test cases the number of difference between the number of edges and the number of nodes would always be equal to negative one (-1) since a test case a single traverse from the system's control flow, and the number of edges would always be lower than the number of nodes. This would mean that the cyclomatic complexity of every traversed test case is always equal to zero (0), which is highly inaccurate. Thus to more reliably measure the complexity, other variables must be considered for test case complexity.

According to many researches, the longer the test case, the more complicated would be to execute it [36, 51 and 54]. Thus it can be concluded that the one of the most important factors in measuring test case complexity is the number of flows. Depending on whether test cases are being extracted from use cases or control flows the edges of the graph are the flows in between the nodes or stages. However other than the flows, there are number of components and items that exist in a test case. Test items have a direct influence on the test case complexity, the more items that are involved in a test case, the more complex that test case will be. In Test Case Reusability Metrics Model [10], it has been noted that more test items reduce the degree of understandability, and thus are an important part of a test case.

To measure complexity first the purpose of the generated output must be determined. Measuring complexity would mean that the higher the percentage of complexity, the more complex it is. This won't fit the other criteria since factors such as understandability, independence, changeability and universal are more reusable if the value is closer to 100%, however complexity would be more reusable if the value is closer to 0%. This means that in order to properly measure reusability the factor that needs to be measured is "Reversed Complexity", which in this research is named "Simplicity" for convenience.

Cyclomatic Complexity grants a basic idea of the need to measure complexity from a white box perspective. However according to researches performed in the field of measuring

complexity [54, 56, 57 and 59], it has been suggested that a hybrid solution of both black box and white box methods is required in order to accurately measure the complexity of a component, and thus in order to measure the complexity of the test case two aspects are used that comprise of a grey box method.

Simplicity is based on two components, test case flows and test items. This means that longer test cases are more complex as they take more time to execute. And the more items that are in the focus of a single test case, the harder it will be and thus more complex. To measure these factors we follow the same concepts from cyclomatic complexity [54] and test case reusability metrics model [10]. In order to measure test case flow simplicity, consider “ $S_E(t)$ ” to be the simplicity of test case “ t ” based on test case flow. And consider “ N_E ” being the number of flows and edges in a control flow or a test case. Thus:

$$S_E(t) = \frac{1}{N_E}$$

Note that the minimum number of flows is one (1) and the minimum number of states or nodes is two (2). Calculating based on the test cases from Figure 3-1 would result in the following table:

Table 3-1: Test Case Flow Simplicity

$TC_1 = \{s_1, s_2\} \rightarrow S_E(TC_1) = 1$	$TC_8 = \{s_1, s_4, s_3, s_5\} \rightarrow S_E(TC_8) = \frac{1}{3}$
$TC_2 = \{s_1, s_3\} \rightarrow S_E(TC_2) = 1$	$TC_9 = \{s_2, s_3\} \rightarrow S_E(TC_9) = 1$
$TC_3 = \{s_1, s_4\} \rightarrow S_E(TC_3) = 1$	$TC_{10} = \{s_2, s_3, s_5\} \rightarrow S_E(TC_{10}) = \frac{1}{2}$
$TC_4 = \{s_1, s_2, s_3\} \rightarrow S_E(TC_4) = \frac{1}{2}$	$TC_{11} = \{s_3, s_5\} \rightarrow S_E(TC_{11}) = 1$
$TC_5 = \{s_1, s_2, s_5\} \rightarrow S_E(TC_5) = \frac{1}{2}$	$TC_{12} = \{s_4, s_3\} \rightarrow S_E(TC_{12}) = 1$
$TC_6 = \{s_1, s_4, s_3\} \rightarrow S_E(TC_6) = \frac{1}{2}$	$TC_{13} = \{s_4, s_3, s_5\} \rightarrow S_E(TC_{13}) = \frac{1}{2}$
$TC_7 = \{s_1, s_2, s_3, s_5\} \rightarrow S_E(TC_7) = \frac{1}{3}$	

In order to calculate simplicity based on test items a similar approach is followed, in which the number of items being tested in a single test case is represented by “ N_{TI} ” and the simplicity of test case items of test case “ t ” is shown as “ $S_{TI}(t)$ ”. The formula is:

$$S_{TI}(t) = \frac{1}{N_{TI}}$$

The minimum number of items being tested is always equal to one (1), since at any given moment at least a single item is tested.

The final simplicity of test case “ t ” would be an average of the two values. Consider “ $S(t)$ ” as the total percentage of simplicity in test case “ t ”. The following is the result of calculating both values:

$$S(t) = \frac{S_E(t) + S_{TI}(t)}{2}$$

3. Model for Measuring Test Case Reusability

In order to measure the reusability metrics for the test case “t”, a combination of all the accepted and developed factors is required. The final two factors are:

1. Independence:

$$I(t) = 2^{-p} = \frac{1}{2^p}$$

2. Simplicity:

$$S_F(t) = \frac{1}{N_E} \text{ and } S_I(t) = \frac{1}{N_{TI}}$$

$$S(t) = \frac{S_E(t) + S_{TI}(t)}{2}$$

The final reusability measurement model is an average of the two main factors; consider “R(t)” as reusability percentage of test case “t”:

$$R(t) = \frac{I(t) + S(t)}{2}$$

4. CHAPTER 4: EVALUATION

A. Method of Evaluation

To evaluate the results of test case reusability measurement model, a process is used which is based on several researches related to generating test cases for reuse and other purposes.

The first important note is mentioned in the research “A new Approach to Generating High Quality Test Cases” [17]. They explain that in order to have a high quality test case one must cover all aspects that a fault can occur. A process is then made to test every aspect of a process – this also aids in measuring the independence of each process and how it is related and linked to other test cases. Hence in Figure 4-1 we can see all the workings of the described login test case based on all included processes:

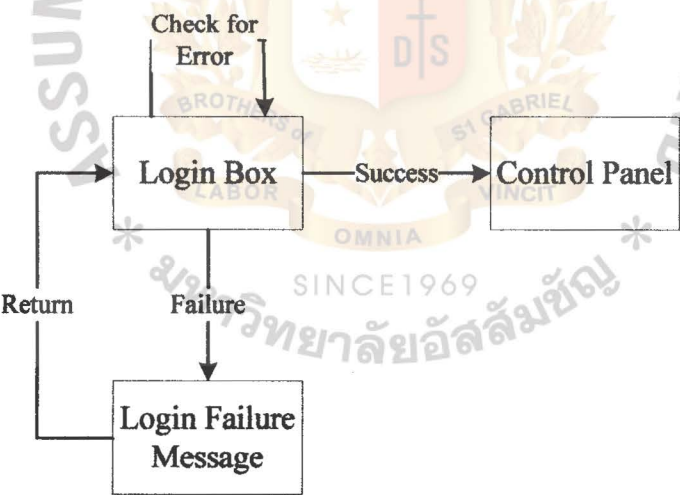


Figure 4-1: Login Process Test Flow [17]

However the main steps of the test case generation process is based on the IBM Rational Edge [26] on generating test cases from use cases. This method provides a number of steps which are malleable and can be used in not only generating test cases from use cases, but also

generating the test cases from other mediums as well, such as control flows and other sources. The steps are as following:

1. Step 1: Generate Use Cases [26] (or Control Flows).
2. Step 2: Generate Use Case Scenarios [26] (or Control Flow Traverses).
3. Step 3: Identify Test Cases [26].
4. Step 4: Identify Data Values to Test [26].
5. Step 5: Apply Reusability Measurement Model.
6. Step 6: Collect Generated Results.

Each step is applied depending on what method of extraction is used. When all test cases are completely generated, then the reusability measurement model is applied and the results are collected based on Independence, and Simplicity. In the end the two factors are put together and calculated as a single value for measuring test case reusability.

B. Evaluation Samples

In order to properly evaluate the results three samples are used from separates researches and projects. Each uses a different method of test case generation which will provide an insight into the feasibility of the system. The three researches are:

1. IBM Rational Edge – Generating Test Cases from Use Cases [26]: a student registration system with accessible use cases, scenarios and test cases.
2. Hotel Management System [23]: a fully accessible system which is designed for the purpose of testing. The only component that is used is a login for the guest.
3. Research on “An Enhanced Test Case Generation Technique Based on Activity Diagrams” [55] – test cases that are generated based on activity diagrams.

Each of the examples will be evaluated based on the 6 steps of evaluation. However since different techniques are used the content of some steps is changed however the concept remains intact.

1. Course Registration System

In the IBM Rationale Edge “Generating Test Cases from Use Cases” [26], a course registration system is created in which based on the use case diagram test cases are generated.

Figure 4-7 the use case for the registration system is demonstrated by the author of the article.

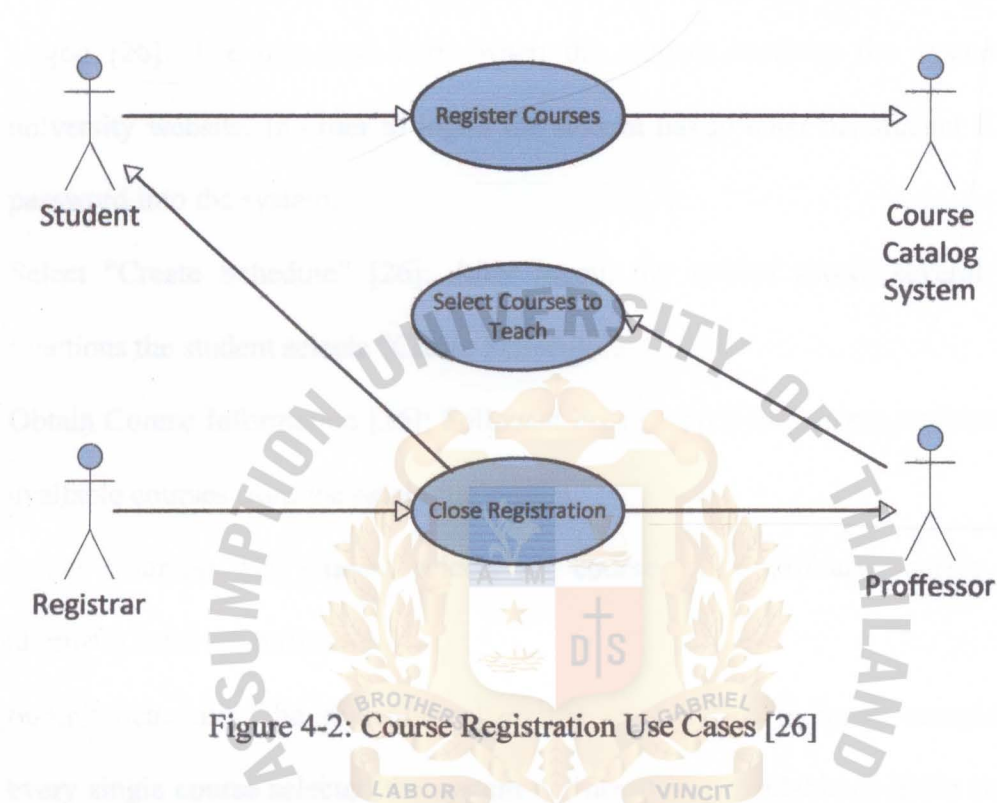


Figure 4-2: Course Registration Use Cases [26]

There are three use cases in Figure 4-2. They are Register Course, Select Courses to Teach and Close Registration. All the three use cases interrelate in one way or another and affect each other depending on the phase and sub phase of each test case.

Following the steps to generate test cases, the next step is to generate use case scenarios and create test case based on the scenarios and control flows. The use case has a basic flow and five alternate flows which combined create the different scenarios to generate the test cases. The basic flow consists of five steps. Those steps are [26]:

1. Logon [26]: The use case starts when the student accesses the system via the university website. In order to logon the student has to enter his student ID and the password into the system.
2. Select “Create Schedule” [26]: After logon, the system shows several available functions the student selects “Create Schedule”.
3. Obtain Course Information [26]: Followed by the step 2 the system retrieves a list of available courses from the catalogue system.
4. Select Courses: The student selects six courses (four primary courses and two alternate) from the retrieved list.
5. Submit Schedule: The student finalizes the selected and indicates completion. For every single course selected the system verifies if it is available or if the student has passed the necessary prerequisites.
6. Display Completed Schedule: In the final step if the submission was successful the system displays the schedule containing the selected courses for the student and a confirmation number for the schedule.

The basic flow is when the process of the control flow is straight and all the decision points have valid variables. The alternate flow is possible points that appear as decision points within the system and depending on the status of the values or the condition of the system overall they produce an alternative path.

The alternative flows in the course registering use case are:

1. Unidentified student [26]: In step 1 of the basic flow (logon) if the credentials inserted (namely the student id and/or the password) are not valid an error message is displayed and notifies the student of the status.
2. Quit [26]: At any point of the system the user can quit and exit the system. By this point the use case ends.
3. Unfulfilled Prerequisites, Course Full or Schedule Conflicts [26]: In step 5 of the basic flow (submit schedule) if the student has unfulfilled prerequisites or if the courses are full or if there are schedule conflicts within the selection the system displays a message that the student should select a different course and it directs to continue at step 4 which is the selection of courses.
4. Course Catalogue System Unavailable [26]: In step 3 of the basic flow (Obtain Course Information) if the system is unavailable a message is displayed and the use case ends.
5. Course Registration Closed [26]: The use case ends and a message is displayed if it is determined that the registration is closed at any point of the system.

Using the information provided in the basic flow and the alternate flow a control flow is designed that reflects all the possible paths. From which the test cases are then generated.

Figure 4-3 is the control flow of the system based on the provided information.

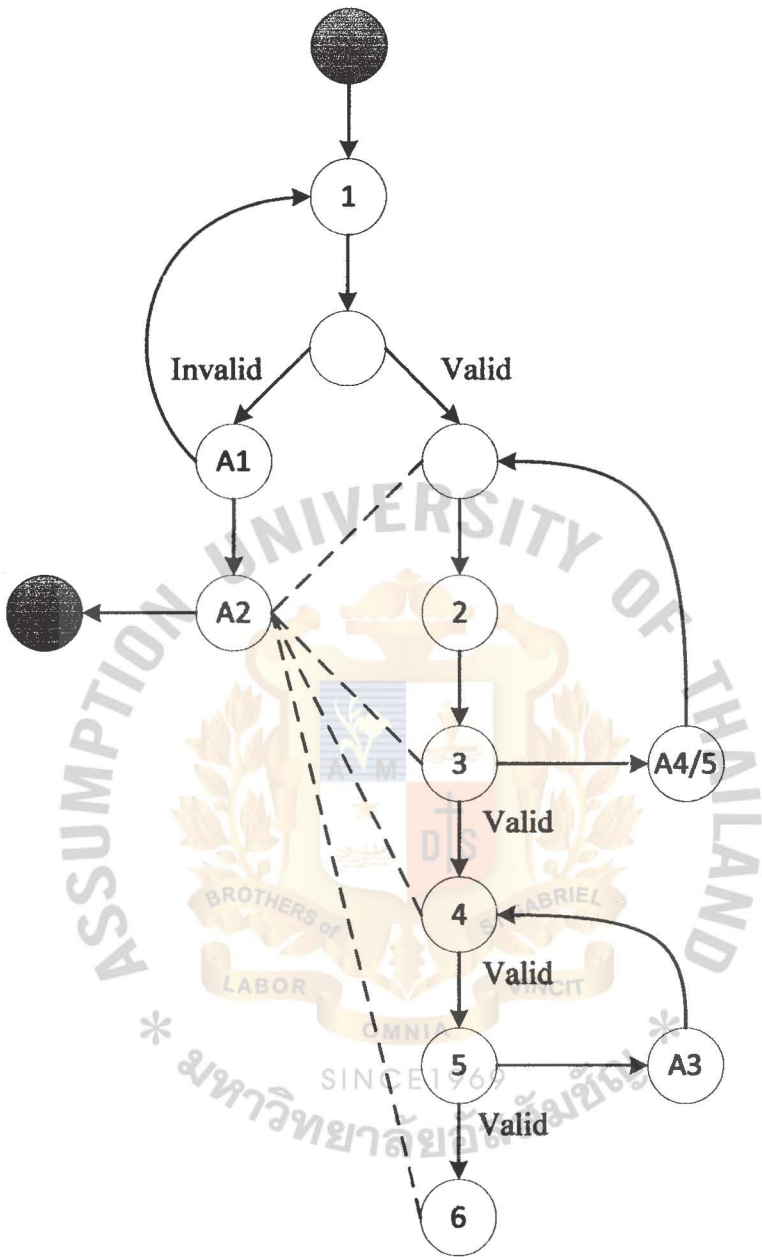


Figure 4-3: Course Register Control Flow

According to the developers there are 8 possible scenarios to test the use case in its entirety.

Those scenarios are listed as follows:

1. Successful Registration [26].
2. Unidentified Student [26].
3. Valid User Quits [26].
4. Course Registration System Unavailable [26].
5. Registration Closed [26].
6. Cannot Enroll – Course Full [26].
7. Cannot Enroll – Prerequisite Not Fulfilled [26].
8. Cannot Enroll – Schedule Conflict [26].

The test cases for the Course Register use case will mainly be Test Case Value Matrix [26].

This contains detailed information on the values and the expected result. Other information is not added and not seemed necessary by the author.

However there are other test cases that exist apart from the 8 main test cases. These are the sub test cases that are generated for the alternate 2 which point out that at any given point the user can quit. So this results in three additional test cases.

Table 4-1: "Course Register" Test Case Matrix

Test Case ID	Test Case Objective/Scenario	Student ID	Password	Courses Selected	Prerequisite Fulfilled	Course Open	Schedule Open	Expected Result
RC_1	Successful Registration	Valid	Valid	Valid	Valid	Valid	Valid	Schedule and confirmation number displayed.
RC_2	Unidentified Student	Invalid	N/A	N/A	N/A	N/A	N/A	Error Message; back to login screen
RC_3	Valid User Quits	Valid	Valid	N/A	N/A	N/A	N/A	Login Screen Appears
RC_4	Course Registration System Unavailable	Valid	Valid	N/A	N/A	N/A	N/A	Error Message; Back to step 2
RC_5	Registration Closed	Valid	Valid	N/A	N/A	N/A	N/A	Error Message; Back to step 2
RC_6	Cannot Enroll – Course Full	Valid	Valid	Valid	Valid	Invalid	Valid	Error Message; Back to step 3
RC_7	Cannot Enroll – Prerequisite Not Fulfilled	Valid	Valid	Valid	Invalid	Valid	Valid	Error Message; Back to step 4
RC_8	Cannot Enroll – Schedule Conflict	Valid	Valid	Valid	Valid	Valid	Invalid	Error Message; Back to step 4

Table 4-1 lists all the test cases and the values at that phase. Table 4-12 lists all the precursor tests and involved test items.

Table 4-2: Register Course Test Links and Test Items

Test Case ID	Independence	Test Items	Number of Test Items	Number of Edges
RC_1	0	1, 2, 3, 4, 5, 6	6	8
RC_2	0	1	1	3
RC_3	0	1, A2	2	5
RC_4	0	1, A4	2	5
RC_5	0	1, A5	2	5
RC_6	0	1, 2, 3, 4, 5, A3	6	9
RC_7	0	1, 2, 3, 4, 5, A3	6	9
RC_8	0	1, 2, 3, 4, 5, A3	6	9
RC_9	1	3, A2	2	3
RC_10	1	4, A2	2	3
RC_11	1	6, A2	2	3

Using the values from the test cases we calculate the reusability of the test case by first calculating independence and then simplicity and finally the overall reusability.

In order to measure independence we use the values in Table 4-3.

Table 4-3: Course Register Test Case Independence

Test Case ID	Independence	Independence Percentage
RC_1	$I(t) = 2^{-0} = 1$	100%
RC_2	$I(t) = 2^{-0} = 1$	100%
RC_3	$I(t) = 2^{-0} = 1$	100%
RC_4	$I(t) = 2^{-0} = 1$	100%
RC_5	$I(t) = 2^{-0} = 1$	100%
RC_6	$I(t) = 2^{-0} = 1$	100%
RC_7	$I(t) = 2^{-0} = 1$	100%
RC_8	$I(t) = 2^{-0} = 1$	100%
RC_9	$I(t) = 2^{-1} = 0.5$	50%
RC_10	$I(t) = 2^{-1} = 0.5$	50%
RC_11	$I(t) = 2^{-1} = 0.5$	50%

To measure simplicity, the values from Table 4-4 are extracted. The test items are the involved items in the test cases. The test edges are the number of edges that are involved in every test case.

Table 4-4: Simplicity Measurement Results

Test Case ID	Test Item Simplicity	Test Edge Simplicity	Simplicity Percentage
RC_1	$S_{TI}(t) = \frac{1}{N_{TI}} = \frac{1}{6}$	$S_E(t) = \frac{1}{N_E} = \frac{1}{8}$	14.58%
RC_2	$S_{TI}(t) = \frac{1}{N_{TI}} = 1$	$S_E(t) = \frac{1}{N_E} = \frac{1}{3}$	66.66%
RC_3	$S_{TI}(t) = \frac{1}{N_{TI}} = \frac{1}{2}$	$S_E(t) = \frac{1}{N_E} = \frac{1}{5}$	35%
RC_4	$S_{TI}(t) = \frac{1}{N_{TI}} = \frac{1}{2}$	$S_E(t) = \frac{1}{N_E} = \frac{1}{5}$	35%
RC_5	$S_{TI}(t) = \frac{1}{N_{TI}} = \frac{1}{2}$	$S_E(t) = \frac{1}{N_E} = \frac{1}{5}$	35%
RC_6	$S_{TI}(t) = \frac{1}{N_{TI}} = \frac{1}{6}$	$S_E(t) = \frac{1}{N_E} = \frac{1}{9}$	*27.77%
RC_7	$S_{TI}(t) = \frac{1}{N_{TI}} = \frac{1}{6}$	$S_E(t) = \frac{1}{N_E} = \frac{1}{9}$	27.77%
RC_8	$S_{TI}(t) = \frac{1}{N_{TI}} = \frac{1}{6}$	$S_E(t) = \frac{1}{N_E} = \frac{1}{9}$	27.77%
RC_9	$S_{TI}(t) = \frac{1}{N_{TI}} = \frac{1}{2}$	$S_E(t) = \frac{1}{N_E} = \frac{1}{3}$	41.66%
RC_10	$S_{TI}(t) = \frac{1}{N_{TI}} = \frac{1}{2}$	$S_E(t) = \frac{1}{N_E} = \frac{1}{3}$	41.66%
RC_11	$S_{TI}(t) = \frac{1}{N_{TI}} = \frac{1}{2}$	$S_E(t) = \frac{1}{N_E} = \frac{1}{3}$	41.66%

By combining the two tables the overall reusability based on simplicity and independence is acquired. All the values are included in the Table 4-5.

Table 4-5: Course Register Test Case Reusability

Test Case ID	Independence	Simplicity	Reusability Percent
RC_1	100.0%	14.58%	57.3%
RC_2	100.0%	66.66%	83.3%
RC_3	100.0%	35%	67.5%
RC_4	100.0%	35%	67.5%
RC_5	100.0%	35%	67.5%
RC_6	100.0%	27.77%	63.9%
RC_7	100.0%	27.77%	63.9%
RC_8	100.0%	27.77%	63.9%
RC_9	50.0%	41.66%	45.8%
RC_10	50.0%	41.66%	45.8%
RC_11	50.0%	41.66%	45.8%



2. The ATM Withdraw Based on Activity Diagram

In the research “An Enhanced Test Case Generation Technique Based on Activity Diagrams” [55], there is a test case generation technique which extracts test cases based on Activity Diagrams. One unique aspect of this research is that the basis of the test case generation method is cyclomatic complexity [54], which allows them measure the number of test paths necessary for a full coverage. However for the purpose of evaluating test cases based on reusability, the methodology is not the main concentration, but the generated results are.

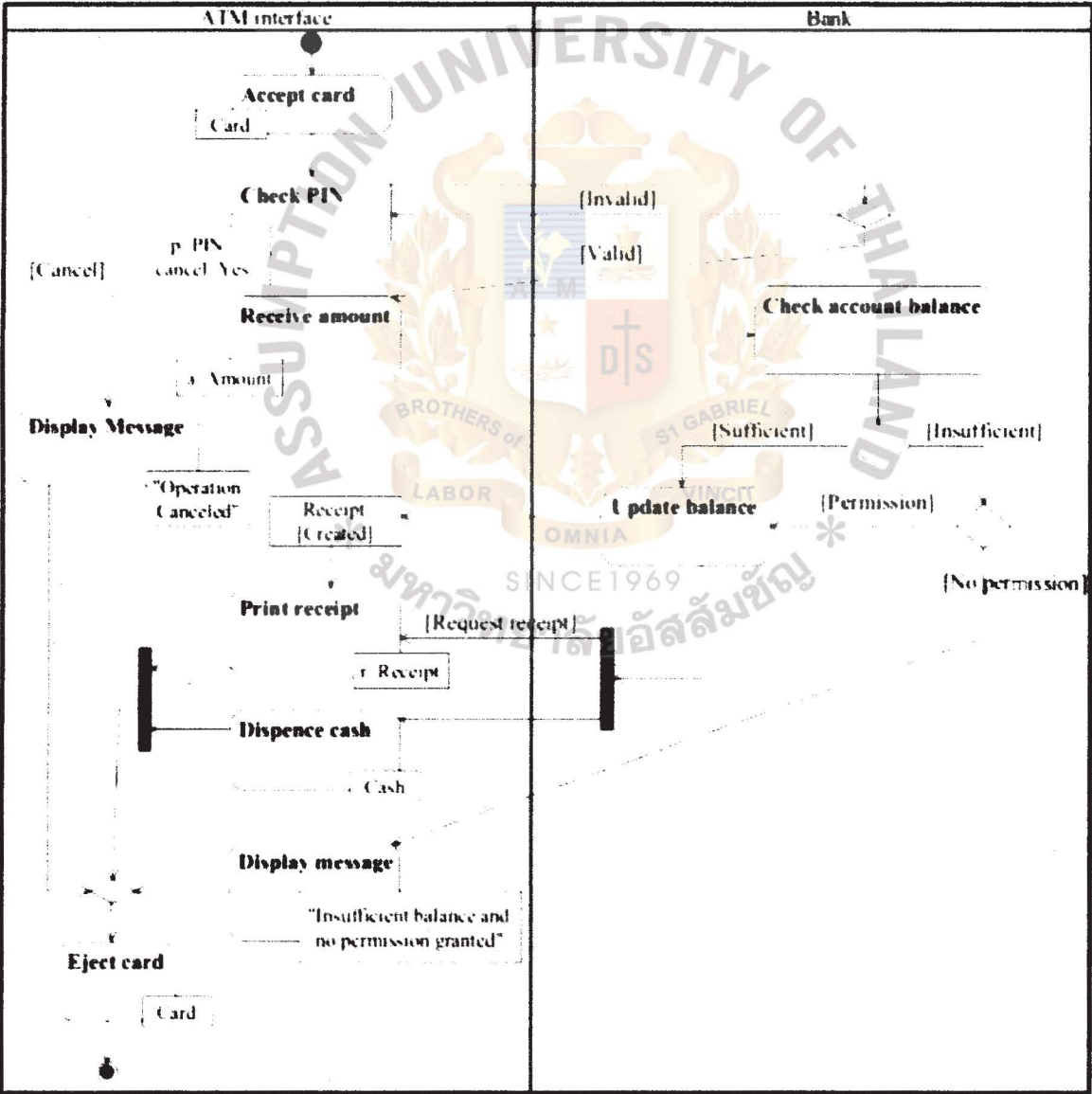


Figure 4-4: ATM Withdraw Activity Diagram [55]

The “ATM Withdraw” is the example used to demonstrate their model, in which we use to evaluate the measurement model. The Activity Diagram for the ATM Withdraw is shown in Figure 4-4.

The system has two main interfaces; one is the ATM interface by which the user interacts with, and the other the Bank interface which the ATM interacts with.

The system starts by accepting the ATM card from the user, once accepted the user proceeds to insert his pin number which is promptly checked by bank, if invalid the user may have to enter the pin again or cancel which then the system will respond by ejecting the card. If the pin is accepted the user then proceeds to enter the required amount, the amount is then checked with the bank and the account holder’s balance, if insufficient the bank then checks if the user has permission to over withdraw from balance, if there is no permission the message “Insufficient balance and no permission granted” will appear on the screen and the card is ejected. If there is permission the balance is then updated and a receipt created. The receipt is then printed and the cash is dispensed at the same time and the system proceeds to eject card and end.

In order to extract test cases from activity diagrams, they must be converted to Activity Diagram Graphs (ADG) which essentially serves as control flows for the system. The converted diagram is created by the researchers of Test Case Generation Based on Activity Diagram [55] and is shown in Figure 4-5.

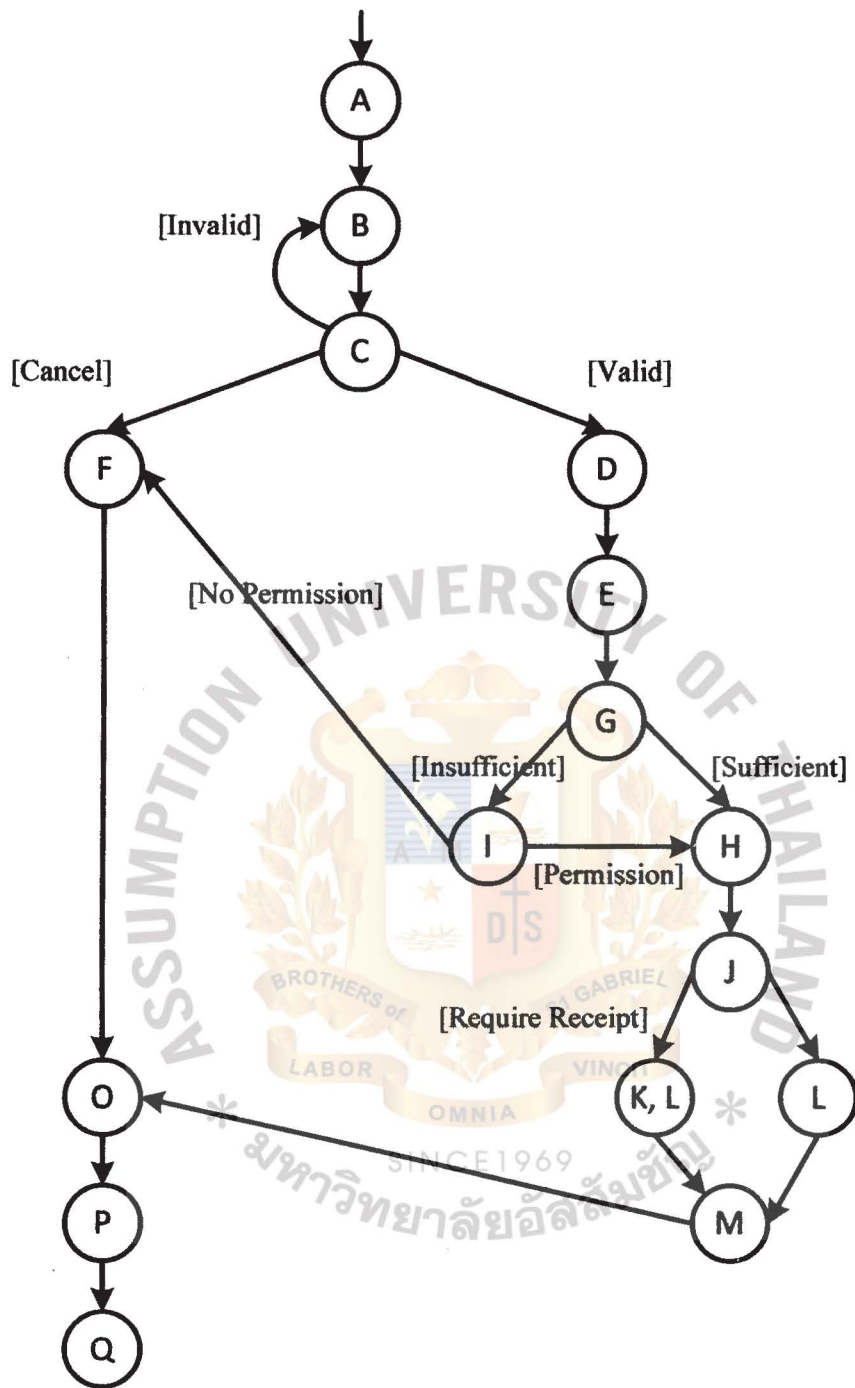


Figure 4-5: ATM Withdraw Activity Diagram Graph and Control Flow [55]

The test paths are shown in Figure 4-6, in which all the paths are included, note that no test case removal technique is used to remove redundant test cases, and test cases are measured in their raw and unmodified form.

<i>Test Path 1</i> = { <i>A</i> → <i>B</i> → <i>C</i> → <i>B</i> }
<i>Test Path 2</i> = { <i>A</i> → <i>B</i> → <i>C</i> → <i>D</i> → <i>E</i> → <i>G</i> → <i>H</i> → <i>J</i> → <i>KL</i> → <i>M</i> → <i>O</i> → <i>P</i> → <i>Q</i> }
<i>Test Path 3</i> = { <i>A</i> → <i>B</i> → <i>C</i> → <i>D</i> → <i>E</i> → <i>G</i> → <i>H</i> → <i>J</i> → <i>L</i> → <i>M</i> → <i>O</i> → <i>P</i> → <i>Q</i> }
<i>Test Path 4</i> = { <i>A</i> → <i>B</i> → <i>C</i> → <i>D</i> → <i>E</i> → <i>G</i> → <i>I</i> → <i>F</i> → <i>O</i> → <i>P</i> → <i>Q</i> }
<i>Test Path 5</i> = { <i>A</i> → <i>B</i> → <i>C</i> → <i>D</i> → <i>E</i> → <i>G</i> → <i>I</i> → <i>H</i> → <i>J</i> → <i>KL</i> → <i>M</i> → <i>O</i> → <i>P</i> → <i>Q</i> }
<i>Test Path 6</i> = { <i>A</i> → <i>B</i> → <i>C</i> → <i>D</i> → <i>E</i> → <i>G</i> → <i>I</i> → <i>H</i> → <i>J</i> → <i>L</i> → <i>M</i> → <i>O</i> → <i>P</i> → <i>Q</i> }
<i>Test Path 7</i> = { <i>A</i> → <i>B</i> → <i>C</i> → <i>F</i> → <i>O</i> → <i>P</i> → <i>Q</i> }

Figure 4-6: Generated Test Paths for ATM Withdraw [55]

Each test path represents a single test case. Each test case required a certain number of values in order to operate; the values are either a variable or a constant depending on the role they possess in the system. Table 4-6 lists all the required values.

Table 4-6: ATM Withdraw Test Case Values

Test Case ID	Pin	Amount	Balance	Permission
ATM_1	PIN [Invalid]	N/A	N/A	N/A
ATM_2	PIN [Valid]	Amount [Sufficient]	≥ Amount	N/A
ATM_3	PIN [Valid]	Amount [Sufficient]	≥ Amount	N/A
ATM_4	PIN [Valid]	Amount [Insufficient]	≤ Amount	No
ATM_5	PIN [Valid]	Amount [Insufficient]	≤ Amount	Yes
ATM_6	PIN [Valid]	Amount [Insufficient]	≤ Amount	Yes
ATM_7	PIN [Invalid]	N/A	N/A	N/A

Table 4-7: ATM Withdraw Test Cases 1, 2 and 3

Test Case ID	Test Item	Test Case Description /Objective	Test Case Keyword	Test Links	Steps to Execute	Expected Result
ATM_01	PIN Box	Invalid PIN - Return to PIN Box	ATM Login, Invalid, Verification	0	<ol style="list-style-type: none"> 1. Enter ATM Card 2. Enter PIN [Invalid] 3. Verify Error 4. Enter PIN [Invalid] 	Invalid PIN Result in Invalid Verification and return to page when prompt.
ATM_02	PIN Box, Amount Box, Balance Check, Receipt Print,	Valid Pin - Sufficient Amount - Dispense Receipt	ATM Login, Valid, Verification, Sufficient Amount, Sufficient Balance, Dispense Receipt	0	<ol style="list-style-type: none"> 1. Enter ATM Card 2. Enter PIN [Valid] 3. Verify Validity 4. Enter Amount [Sufficient] 5. Receive Receipt 6. Receive Card 7. Return 	Print Receipt - Ejected Card
ATM_03	PIN Box, Amount Box, Balance Check, Cash Dispenser	Valid Pin - Sufficient Amount - Dispense Amount	ATM Login, Valid, Verification, Sufficient Amount, Sufficient Balance, Dispense Amount	0	<ol style="list-style-type: none"> 1. Enter ATM Card 2. Enter PIN [Valid] 3. Verify Validity 4. Enter Amount [Sufficient] 5. Receive Dispensed Amount 6. Receive Card 7. Return 	Dispense Cash - Ejected Card

Table 4-8: ATM Withdraw Test Cases 4 and 5

Test Case ID	Test Item	Test Case Description /Objective	Test Case Keyword	Test Links	Steps to Execute	Expected Result
ATM_04	PIN Box, Amount Box, Balance Check, Permission Check, Display Message Box	Valid Pin - Insufficient Amount - Display Error Message	ATM Login, Valid, Verification, Insufficient Amount, Insufficient Balance, No Permission	0	<ol style="list-style-type: none"> 1. Enter ATM Card 2. Enter PIN [Valid] 3. Verify Validity 4. Enter Amount [Insufficient] 5. Verify No Permission. 6. View Denied Message 7. Eject Card 8. Return 	Display Message: "Insufficient balance and no permission granted" - Ejected Card
ATM_05	PIN Box, Amount Box, Balance Check, Receipt Print, Permission Check	Valid Pin - Insufficient Amount - Permission Granted - Receive Receipt	ATM Login, Valid, Verification, Sufficient Amount, Sufficient Balance, Dispense Receipt, Permission Granted	0	<ol style="list-style-type: none"> 1. Enter ATM Card 2. Enter PIN [Valid] 3. Verify Validity 4. Enter Amount [Insufficient] 5. Verify Permission. 6. Receive Receipt 7. Receive Card 8. Return 	Print Receipt - Ejected Card

Table 4-9: ATM Withdraw Test Cases 6 and 7

Test Case ID	Test Item	Test Case Description /Objective	Test Case Keyword	Test Links	Steps to Execute	Expected Result
ATM_06	PIN Box, Amount Box, Balance Check, Cash Dispenser, Permission Check	Valid Pin - Insufficient Amount - Permission Granted - Dispense Amount	ATM Login, Valid, Verification, Sufficient Amount, Sufficient Balance, Dispense Amount, Permission Granted	0	<ol style="list-style-type: none"> 1. Enter ATM Card 2. Enter PIN [Valid] 3. Verify Validity 4. Enter Amount [Insufficient] 5. Verify Permission. 6. Receive Dispensed Amount 7. Receive Card 8. Return 	Dispense Cash - Ejected Card
ATM_07	PIN Box, Cancel Message Box	Invalid PIN - Cancel Operation,	ATM Login, Invalid, Verification, Cancel Operation	0	<ol style="list-style-type: none"> 1. Enter ATM Card 2. Enter PIN [Invalid] 3. Verify Error 4. Cancel Operation 5. View Cancel Message 6. Eject Card 7. Return 	Cancel Message "Operation Canceled" - Ejected Card

Test Cases are formatted in the same fashion that Hotel Management System Test Cases were formatted which is based on a standard Test Case system. All the seven test cases derived from the several traversed paths of the control flow (Figure 4-3) are listed in three separate tables: Table 4-7, Table 4-8 and Table 4-9.

In order to measure independence the value required is the number of precursor test cases. Since the entire precursor tests are zero (0) then the result is considered to be 100% independent. Table 4-10 demonstrates how the calculation is done for each value.

Table 4-10: Independence Measurement Results

Test Case ID	Independence	Independence Percentage
ATM_01	$I(t) = 2^{-0} = 1$	100%
ATM_02	$I(t) = 2^{-0} = 1$	100%
ATM_03	$I(t) = 2^{-0} = 1$	100%
ATM_04	$I(t) = 2^{-0} = 1$	100%
ATM_05	$I(t) = 2^{-0} = 1$	100%
ATM_06	$I(t) = 2^{-0} = 1$	100%
ATM_07	$I(t) = 2^{-0} = 1$	100%

In order to measure Simplicity the number of test items and the number of edges in the graph test path are of main concern. Figure 4-3 includes all the various test paths which can be used to measure test path Simplicity. The result of calculating each phase of simplicity is shown in Table 4-11. Note that the final percentage is equal to the average value of both simplicity factors multiplied by 100.

Table 4-11: Simplicity Measurement Results

Test Case ID	Test Item Simplicity	Test Edge Simplicity	Simplicity Percentage
ATM_01	$S_{TI}(t) = \frac{1}{N_{TI}} = 1$	$S_E(t) = \frac{1}{N_E} = \frac{1}{4}$	62.5%
ATM_02	$S_{TI}(t) = \frac{1}{N_{TI}} = \frac{1}{4}$	$S_E(t) = \frac{1}{N_E} = \frac{1}{13}$	16.35%
ATM_03	$S_{TI}(t) = \frac{1}{N_{TI}} = \frac{1}{4}$	$S_E(t) = \frac{1}{N_E} = \frac{1}{13}$	16.35%
ATM_04	$S_{TI}(t) = \frac{1}{N_{TI}} = \frac{1}{5}$	$S_E(t) = \frac{1}{N_E} = \frac{1}{11}$	14.54%
ATM_05	$S_{TI}(t) = \frac{1}{N_{TI}} = \frac{1}{5}$	$S_E(t) = \frac{1}{N_E} = \frac{1}{14}$	13.57%
ATM_06	$S_{TI}(t) = \frac{1}{N_{TI}} = \frac{1}{5}$	$S_E(t) = \frac{1}{N_E} = \frac{1}{14}$	13.57%
ATM_07	$S_{TI}(t) = \frac{1}{N_{TI}} = \frac{1}{4}$	$S_E(t) = \frac{1}{N_E} = \frac{1}{7}$	32.14%

Using the values calculated in Table 4-10 and Table 4-11 the following table is created and the results are calculated as the average of the two values calculates the overall reusability of a single test case.

$$R(t) = \frac{I(t) + S(t)}{2}$$

Table 4-12: Reusability Measurement Results

Test Case ID	Independence	Simplicity	Reusability Percent
ATM_01	100%	62.50%	81.3%
ATM_02	100%	16.35%	58.2%
ATM_03	100%	16.35%	58.2%
ATM_04	100%	14.54%	57.3%
ATM_05	100%	13.57%	56.8%
ATM_06	100%	13.57%	56.8%
ATM_07	100%	32.14%	66.1%

3. Hotel Management System Guest Login

The Hotel Management System Guest Login was designed and engineered based on the Hotel Management System Login System [23]. The Login System is reengineered in order to be used as an example for measuring how would the Test Case Measurement System behave when faced with rare situations such as transitioning back to previous nodes and how would precursor test case fit in a general scenario of test case production.

The login system is simple, the user insets username and password, and then clicks on login and depending on the validity of the variables two outcomes are expected. A valid response will result in a welcome message and sends the user to the guest menu where he can perform his guest functions (this part of the system is not demonstrated in the control flow). An invalid response would result in an error message which would return the user in entering his username and password again. Figure 4-7 demonstrates the control flow of the guest login.

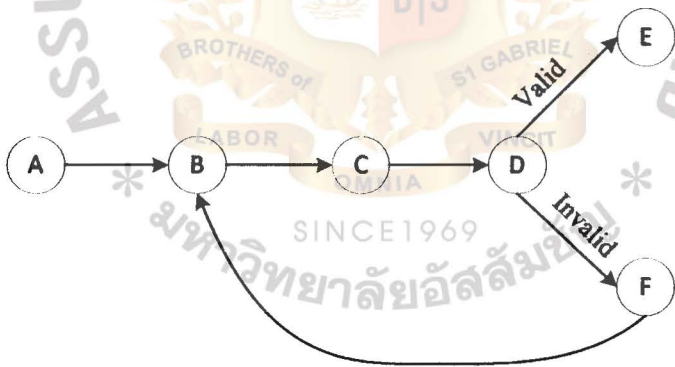


Figure 4-7: HMS GUEST LOGIN Control Flow

Following the steps, each traverse results in a test case, in this scenario we have two main traverses which are shown in Figure 4-8.

<i>Test Path 1 = {A → B → C → D → E}</i>
<i>Test Path 2 = {A → B → C → D → F → B}</i>

Figure 4-8: HMS Guest Login Test Paths

Attributes used are: Test Case ID – a unique attribute to designate each specific test case for later reference. Test Item – the main items that the test case is examining. Test Case Description/Objective – describing the main objective of the test case. Test Case Keyword – related keywords to the particular test case used for referencing and also for reusability measures when they want to reuse test cases that cover a particular area. Test Case Link/Prerequisites – the number of processes that comes before the test case and it is needed to be tested first in order to successfully test that item. Steps to Execute – is basically Test Steps (steps that must be taken in order to reproduce a certain test objective). Expected Result – is the result that is expected to be seen when the test steps are successfully performed.

Test Data – also known as “Data required”, is the data that is needed in order to successfully test the test case. This information is stored in a separate table. Table 4-13 includes all the data required to execute the related test case.

Table 4-13: Test Case Required Data Table

Test Case ID	Username	Password
HM_GL_01	Username [Invalid]	Password [Invalid]
HM_GL_02	Username [Valid]	Password [Valid]

Table 4-14: Sample Test Cases from Hotel management System Login Process

Test Case ID	Test Item	Test Case Description /Objective	Test Case Keyword	Test Links	Steps to Execute	Expected Result
HM_GL_01	Username Box, Password Box, Display Message	Invalid Login - Blank Username and Password	Login, Verification, Invalid, Username, Password, Message Box	0	<ol style="list-style-type: none"> 1. Enter Login Page 2. Input Username. 3. Input Password. 4. Click on Login 5. Confirm Error Message 6. Input Username 	Error Message: Invalid Credentials. Return to Login Page.
HM_GL_02	Username Box, Password Box, Display Message	Valid Login - Correct Username and Password	Login, Verification, Valid, Username, Password, Message Box	0	<ol style="list-style-type: none"> 1. Enter Login Page 2. Input Username. 3. Input Password. 4. Click on Login 5. Confirm Welcome Message 	Welcome Message: Successful Login - Guest Profile

Table 4-14 includes all the test cases that need to be executed. In order to apply the measurement model the following data is required:

For independence the value required to be measured is the number of precursor tests. Since most of the precursor tests are zero (0) then the result is considered to be 100% independent.

Table 4-15 demonstrates how the calculation is done for each value.

Table 4-15: Independence Measurement Results

Test Case ID	Independence	Independence Percentage
HM_GL_01	$I(t) = 2^{-0} = 1$	100.0%
HM_GL_02	$I(t) = 2^{-0} = 1$	100.0%

For Simplicity two main values are of concern. Test Items and Test Steps/Stages/Flows depending on the case used. In this test case we use test steps as indication of measuring test case simplicity.

Table 4-16: Simplicity Measurement Results

Test Case ID	Test Item Simplicity	Test Step Simplicity	Simplicity Percentage
HM_GL_01	$S_{TI}(t) = \frac{1}{N_{TI}} = \frac{1}{3}$	$S_E(t) = \frac{1}{N_E} = \frac{1}{5}$	26.66%
HM_GL_02	$S_{TI}(t) = \frac{1}{N_{TI}} = \frac{1}{3}$	$S_E(t) = \frac{1}{N_E} = \frac{1}{4}$	29.16%

Based on the provided information from Table 4-15 and Table 4-16 the following table is created and the results are calculated as the average of the three values calculates the overall reusability of a single test case.

$$R(t) = \frac{I(t) + S(t)}{2}$$

Table 4-17: Reusability Measurement Results

Test Case ID	Independence	Simplicity	Reusability Percent
HM_GL_01	100.0%	26.66%	63.3%
HM_GL_02	100.0%	29.16%	64.58%



5. CHAPTER 5: RESULTS AND DISCUSSION

To analyze the results the five point linkert scale [40, 41] will be used to measure and identify reusability. The scale is divided evenly into five separate points (Table 5-1):

Table 5-1: Linkert Scale for Test Case Reusability

Percentage Range	Linkert Scale Point
0% – 20%	Not Used
20% – 40%	Difficult to Reuse
40% – 60%	Neither Difficult nor easy to reuse
60% – 80%	Easy to Reuse
80% – 100%	Very Reusable

The scale in Table 5-1 is used to analyze the values obtained via the model for measuring test case reusability. The linkert scale can also be applied in translating the Independence and Simplicity values.

A. Course Registration System

The course registration test cases are based on Rationale Edge from IBM [26]. The results of the measurement for test case reusability result in:

Table 5-2: Course Register Test Case Reusability

Test Case ID	Independence	Simplicity	Reusability Percent
RC_1	100.0%	14.58%	57.3%
RC_2	100.0%	66.66%	83.3%
RC_3	100.0%	35%	67.5%
RC_4	100.0%	35%	67.5%
RC_5	100.0%	35%	67.5%
RC_6	100.0%	27.77%	63.9%
RC_7	100.0%	27.77%	63.9%
RC_8	100.0%	27.77%	63.9%
RC_9	50.0%	41.66%	45.8%
RC_10	50.0%	41.66%	45.8%
RC_11	50.0%	41.66%	45.8%

Test case RC_1 follows the basic flow which is also the longest. It contains 6 items and 8 edges and produces 14.58% simplicity and 100% independence. This indicates that the test case RC_1 is considered not simple in any way. In its current form it can be divided by two sections and separate the login section from the actual registration section, however that would not affect the overall reusability by much since the division of flows would also reduce the independence of the test case to 50%. With an overall reusability of 57.3% is considered to be average, neither easy nor difficult to reuse.

The second test case RC_2 is the shortest alternate flow, with an independence of 100% it is considered fully independent. The simplicity is also measured to be 66.66% with only one

test item and three edges. The overall test case reusability is 83.3% and it is considered very reusable by being between the range of 80% and 100%.

Test cases RC_3, RC_4 and RC_5 have 100% independence and do not start at a middle node. The simplicity is also measured to be 35% with two items and five edges and it is considered not simple. The overall reusability is measured to be 67.5% and it translates “easy to reuse” based on the linkert scale range of over 60%.

Test cases RC_6, RC_7 and RC_8 are have also 100% independence. The simplicity is 27.7% with six items and nine edges. This means that it is not simple. The overall reusability is 63.9% which considered is “easy to reuse”.

Test cases RC_9, RC_10 and RC_11 starts at middle nodes and thus require a precursor test in order to successfully run. Their independence is measure to be 50%. The simplicity is 41.66% with two test items and three edges. The overall reusability is thus measured at 45.8% which based on the linkert scale is considered neither difficult nor easy to reuse.

It is observed that the longer the test case is, the more complex it becomes and thus less simple. Longer test cases also tend to have more items involved, thus reducing the overall simplicity which would lead to less reusability.

B. ATM Withdraw Based on Activity Diagram

The test case paths are based on the control flow diagram traverses which the researchers of the article “Test case Generation Method Based on Activity Diagram” [40]. It has been referred in the research the test cases have not been optimized at all [40] and thus are expected to be average and below average in terms of reuse.

Table 5-3: ATM Withdraw Reusability Measurement Results

Test Case ID	Independence	Simplicity	Reusability Percent
ATM_01	100%	62.50%	81.3%
ATM_02	100%	16.35%	58.2%
ATM_03	100%	16.35%	58.2%
ATM_04	100%	14.54%	57.3%
ATM_05	100%	13.57%	56.8%
ATM_06	100%	13.57%	56.8%
ATM_07	100%	32.14%	66.1%

Test case ATM_01 is considered very reusable mainly due to the value of 81.3% reusability falling in the range of 80% - 100% in the linkert scale. Test cases ATM_02 and ATM_03 have a reusability percentage of 58.2% and thus are considered neither difficult nor easy to reuse. Test cases ATM_04, ATM_05 and ATM_06 also fall into the same category since their values 57.3%, 56.8% and 56.8% respectively are in the scale of neither difficult nor easy to reuse. The last test case (ATM_07) is considered easy to reuse since the value 66.1% is over 60% and below 80%.

Considering all the values generated, it enhanced the main idea that these test cases were not optimized and expected to similar results in case of reusability. Depending on the type of optimization the values may change the overall reusability.

C. Hotel Management System Guest Login

The test cases in HMS were designed based on the Guest Login Sub System. Based on the use cases and system requirements two test cases were created to test all aspects of the Guest Login. The test cases each were measured using the model for measuring test case reusability which consisted of two main criteria: Independence and Simplicity. The results are demonstrated in Table 5-4.

Table 5-4: HMS_GL Test Case Reusability Results

Test Case ID	Independence	Simplicity	Reusability Percent
HM_GL_01	100.0%	26.66%	63.3%
HM_GL_02	100.0%	29.16%	64.58%

The results indicate that test case HM_GL_01 is easy to reuse since the value 63.3% falls in between 60%-80% of the linkert scale. Test case HM_GL_02 is also considered easy to reuse since the value of 64.58%.

D. Test Case Reusability Metrics Model Comparison

In order to evaluate the results a comparison is needed compared to the previous system. However after further examination it is observed that the comparison may not be possible due to the following reasons:

1. Understandability was demonstrated by proof of example and logic that it is unreliable to measure any form of reusability and thus its value cannot be considered to measure test case reusability.
2. Universal requires information that is vague and unidentified. These values are not included in most test case generation techniques and would mean that the TCRMM is only suitable for certain generation methods.
3. Changeability formula does not reflect on the theory provided and thus any measurement on that part would not have been accurate in any form.
4. After further analysis it is seen that Test Case Reusability Metrics Model can only be used to successfully measure metrics from a certain test cases that are generated using “Reusable Test Models and Application Based on Z Specification” [7] as main research which is developed by the same researchers that devised the Test Case Reusability Metrics Model [10].

Following the general concept, we cannot measure any of the current test cases with the TCRMM.

6. CHAPTER 6: CONCLUSION AND FURTHER WORK

A. Conclusion

In the early stages of the research many factors were discovered that had an effect on measuring test case reusability. A Study of Reusability, Complexity and Reuse Design Principles [36] introduced overall Understandability and Complexity as main factors of measuring reusability. The researchers interpreted Understandability as how well defined the interfaces were, and how clear the components were separately on their own without any link to the other components. However the method used for measuring Understandability was via surveys and human perception, meaning that the degree of understandability was measured via how the testers presumed it was and not based on solid measurement model.

The Complexity factor was mentioned in several other researches [36, 51, 54, 57 and 58] as a prominent factor that affects reusability. This factor was later explored and expanded upon in the solutions section and following previous research and calculation methods such as Cyclomatic Complexity [54] and Hybrid Complexity measurement models [57, 58] influenced the direction of which this research used to measure Simplicity (which is defined as reverse complexity).

Other factors such as Changeability, Universal and Independence were elaborated and measured in Test Case Reusability Metrics Model [10], but some such as Changeability and Universal were dismissed due to the fact that they were limited to the scope of their research and test case generation method, and they were not compatible with the objectives and scope of this research.

When the potential factors of Reusability were identified, two of the prominent factors were chosen and expanded upon for use in measuring test case reusability. A template metrics

system was generated that could measure test case reusability of test cases that are generated only in a certain way. The template uses two factors of independence and simplicity as the main criteria for measuring test case reusability. These two factors are considered as a base for the template, and in later editions new factors could be introduced to the system in order to make it more reliable and precise.

In order to evaluate the results of the test case reusability measurement model three main test case samples were used. The first sample is from IBM Rationale Edge article that demonstrates how test cases are generated from use cases and use case scenarios. The results yielded an average result which correlated with the base that all samples that are used indiscriminately are without any optimization and thus expected to yield an average or below average result. The actual results varied between the numbers 40% and 60% which according to the linkert scale translated to average with the exception of one test case that yielded a higher result of 86% which could have been lower if other factors are introduced. The results will get more accurate to the expected results as more factors are added to the template, and also if a weighing system is introduced among the criteria.

The second sample Hotel Management System was used as a basic introductory system to evaluate how the system would behave when multiple returns are introduced. There is also an experimentation of how precursor test cases function.

The third sample ATM Withdraw system test cases were also not optimized and were expected to produce an average result. This was confirmed to be the case when the average of the reusability was somewhere between 50% and 70% which is considered average and above average. In IBM Rationale example for Register Courses the results vary depending on the length of each test case. It has been seen that longer test cases have lower reusability.

B. Drawbacks

Although relatively accurate, the metrics model suffers from some drawbacks that need to be addressed. The drawbacks are as following:

1. Lack of Metrics Weight

There is no weight system in place for the metrics model. In the current model for measuring reusability all the metrics are considered equal in value and weight. This can be rather problematic when a certain organization or developer wants to have a higher emphasis on a certain criterion or factor. Such would be a higher emphasis on simplicity instead of independence. The basic template for the formula is to have a definable weight for every point. However a system must be defined in order to give proper weight to the variables. An example would be:

$$R(t) = \frac{x.I(t) + y.S(t)}{x + y}$$

In the above formula x and y are the weights of each factor. However there currently is no method for assigning those values in an accurate manner.

2. Additional Factors

Currently the formula consists of two main criteria and this would make it inaccurate in most cases. This is particularly the case where the simplicity is very low (closer to 10%) and independence is 100%. This would average to 55% and thus consider it an average reusable test case. Although it may very well be, if there were additional variables, the number could have been more precise.

3. Limitations Due to Scope of Work

Currently the metrics model required the tester to have background knowledge from the test case generation process. This is mainly because of the white box natures that exist within the simplicity measurement factor. In order to design the test cases the tester needs to know about the Control Flows and Use Case Scenarios which indicate the innate complexity of the test case. Other than the white box requirement, there is also a need for Black Box attributes that would aid in a more precise measurement of simplicity according to the hybrid complexity theories mentioned in several researches.

This limitation prevents the model from being used in order to measure automated test cases that do not have a black box aspect, and also it will prevent the use of test cases that are generated without a control flow or that are from a generation method which requires different set of attributed that are not identified in the measurement model.

C. Further Study

This research serves as a template for expanded work on a field that is currently rarely worked upon. There are very few that concentrate on the aspect of singular test case reusability and fewer tend to look for methods of quantifying it. In later researches the drawbacks could be explored more.

New criteria such as changeability and understandability have a higher potential to be considered for the reusability metrics. Both criteria are included in Test Case Reusability Metrics Model, but they are measured based on factors that are not accurate and produce results that are not reliable.



7. REFERENCES AND BIBLIOGRAPHY

- [1] Institute of Electrical & Electronics Engineers, Standard 1059 (1993), IEEE Guide for Software Verification and Validation Plans. 1993.
- [2] S. Schach, Software Engineering, Aksen Associates, Boston, MA, 1990.
- [3] Antonia Bertolino and Eda Marchetti, Software Engineering Body of Knowledge (SWEBOK), 2004 edition, Chapter 5.
- [4] Gregg Rothermel and Mary Jean Harrold. "A Safe, Efficient Algorithm for Regression Test Selection" , 1999
- [5] Institute of Electrical & Electronics Engineers, Standard 610 (1990), reprinted in IEEE Standards Collection: Software Engineering 1994 Edition.
- [6] Cem Kaner, J.D., Ph.D., "What Is a Good Test Case?," pp. 4-5, May 2003.
- [7] Lizhi Cai, Weiqin Tong, Genxing Yang, Zhenyu Liu. "Reusable Test Models and Application Based on Z Specification", pp 4. July, 2007.
- [8] K.K Aggarwal & Yogesh Singh, "Software Engineering", 3rd Edition, 2007, Chapter 9.
- [9] Gregg Rothermel and Mary Jean Harrold. "Analyzing Regression Test Selection Techniques", 1996
- [10] Zhang Juan, Cai Lizhi, Tong Weiqing, Yuan Song, Li Ying, "Test Case Reusability Metrics Model", 2010 2nd International Conference on Computer Technology and Development (ICCTD), November, 2010.
- [11] Huang, J.C. . "Measuring the effectiveness of a test case". appear on 1998 IEEE Workshop on Application-Specific Software Engineering Technology, 1998. ASSET-98. Proceedings. pp 157-159. March, 1998.
- [12] Christian Pfaller, Stefan Wagner, Jorg Gericke, Matthias Wiemann "Multi-Dimensional Measures for Test Case Quality", April, 2008.

- [13] William Frakes, Carol Terry. Software reuse: metrics and models. ACM Computing Surveys, Vol. 28, No.2, June 1996.
- [14] ISO/IEC 9126 Software engineering Product quality, www.jtc1sc7.org
- [15] Jin-Cherng Lin, Kuo-Chiang Wu. "A Model for Measuring Software Understandability". 2006.
- [16] Aggarwal KK, Singh Y., and Chhabra J K , An Integrated Measure of Software Maintainability, Proceedings of Reliability and Maintainability, 2002: 235-241
- [17] Pan Liu, Huaikou Miao. "A New Approach to Generating High Quality Test Cases". 2010.
- [18] Baldwin, C. Y. & Clark, K. B. Design rules, Volume 1: The power of modularity, Cambridge, MA: MIT Press. 2000.
- [19] Douglas D. Lonngren. Reducing the cost of test through reuse. AUTOTESTCON98. IEEE Systems Readiness Technology Conference, 1998 IEEE.
- [20] A von Mayrhauser, R.T.Mraz, J.Walls, and P Ocken. "Domain based testing: Increasing test case reuse". In International Conference on Computer Design, pages 484-491, 1994.
- [21] H. E. Hornstein. Test reuse in cbse using built-in tests. In Proc of the Workshop on Component-Based Software Engineering, Composing Ssystems from Components, Los Alamitos, CA, 2002. IEEE Computer Society Press.
- [22] R. T. Mraz. Domain based testing: A reuse oriented test method. Technical report, colorado state university, 1993.
- [23] Mohammad Rava. "Hotel Management System", Assumption University, 2011.
- [24] WebInject. <http://webinject.org/> . November 29, 2012.
- [25] Russell Turpin, "A Progressive Software Development Lifecycle", Engineering of Complex Computer Systems, 1996.
- [26] Jim Heumann, "Generating Test Cases from Use Cases", The Rational Edge, 2001.
- [27] Alistair Cockburn, "Writing Effective Use Cases", Addison-Wesley, 2001.

- [28] IBM, "Rational Unified Process", Rational Software White Paper, 2001.
- [29] IEEE 829-1998 - Software Quality Engineering - Test Case Specification Template - Version 7.0 - 2001
- [30] R. van Ommering, "Software reuse in product populations," IEEE Transactions on Software Engineering, vol. 31, pp. 537-550, 2005.
- [31] P. Mohagheghi and R. Conradi, "Quality, productivity and economic benefits of software reuse: a review of industrial studies," Empirical Software Engineering, vol. 12, pp. 471-516, 2007.
- [32] P. Mohagheghi and R. Conradi, "An empirical investigation of software reuse benefits in a large telecom product," ACM Transactions on Software Engineering Methodology, vol. 17, pp. 1-31, 2008.
- [33] W. B. Frakes and G. Succi, "An industrial study of reuse, quality, and productivity," Journal of Systems and Software, vol. 57, pp. 99-106, 2001.
- [34] M. Morisio, et al., "Success and Failure Factors in Software Reuse," IEEE Transactions on Software Engineering, vol. 28, pp. 340-357, 2002.
- [35] W. C. Lim, "Effects of Reuse on Quality, Productivity, and Economics," IEEE Softw., vol. 11, pp. 23-30, 1994.
- [36] R. Anguswamy, W. B. Frakes, "A Study of Reusability, Complexity, and Reuse Design Principles". ESEM'12, September 19-20, 2012.
- [37] M. D. McIlroy, et al., "Mass produced software components," Software Engineering Concepts and Techniques, pp. 88-98, 1969.
- [38] Frakes W. Systematic. "Software Reuse: A Paradigm Shift". In Proceedings of Third International Conference on Software Reuse: Advances in Software Reuse. Los Alamitos, California: IEEE Computer Society Press, 1994

- [39] Shaojie Guo, Weiqin Tong, Juan Zhang, Zongheng Liu. "An Application of Ontology to Test Case Reuse". International Conference on Mechatronic Science, Electric Engineering and Computer August 19-22, 2011.
- [40] L. Carmichael, T. Damarla, P McHugh, M. J. Chug. "Issues Involved in Reuse Library for Design for Test". IEEE 1995.
- [41] S. R. Nidumolu and G. W. Knotts, "The effects of customizability and reusability on perceived process and competitive performance of software firms," MIS Q., vol. 22, pp. 105-137, 1998.
- [42] R. Anguswamy and W. B. Frakes, "An Exploratory Study of One-Use and Reusable Software Components," 24th International Conference of Software Engineering and Knowledge Engineering, SEKE'12, San Francisco, USA, 2012.
- [43] W. B. Frakes and R. Baeza-Yates, Information retrieval: data structures and algorithms, 2nd ed. vol. 77. Englewood Cliffs, NJ: Prentice-Hall. , 1998.
- [44] J. Sametingar, Software engineering with reusable components. Berlin Heidelberg, Germany: Springer Verlag, 1997.
- [45] Y. Matsumoto, "Some Experiences in Promoting Reusable Software: Presentation in Higher Abstract Levels," IEEE Transactions on Software Engineering, vol. SE-10, 1984.
- [46] B. Boehm, et al., "Cost estimation with COCOMO II," ed: Upper Saddle River, NJ: Prentice-Hall, 2000.
- [47] N. E. Fenton and M. Neil, "Software metrics: roadmap," presented at the Proceedings of the Conference on The Future of Software Engineering, Limerick, Ireland, 2000.
- [48] R. W. Selby, "Enabling reuse-based software development of largescale systems," IEEE Transactions on Software Engineering, vol. 31, pp. 495-510, 2005.
- [49] A. Gupta, "The profile of software changes in reused vs. non-reused industrial software systems," Doctoral Thesis, NTNU, Singapore, 2009.

- [50] T. Tan, et al., "Productivity trends in incremental and iterative software development," in ESEM '09 Proceedings of the 2009 3rd International Symposium on Empirical Software Engineering and Measurement Lake Buena Vista, Florida, USA, 2009, pp. 1-10.
- [51] S. Roongruangsuwan, J. Daengdej. "Test Case reduction Methods by Using CBR". Autonomous System Research Laboratory. 2010.
- [52] Bruce H. Barnes, Terry B. Bollinger. "Making Reuse Cost Effective". IEEE Software Vol: 8. Issue: 1. Jan. 1991.
- [53] Barry Smyth & Keane. "Remembering To Forget: A Competence Preserving Deletion Policy for Case-Based Reasoning Systems" In Proceedings of the 14th International Joint Conference on Artificial Intelligence, 377-382. Morgan-Kaufman, 1995.
- [54] Thomas J. McCabe. "A Complexity Measure". IEEE Transactions on Software Engineering, Vol. Se-2, No. 4, December 1976.
- [55] Pakinam N. Boghdady, Nagwa L. Badr, Mohamed A. Hashim, Mohamed F. Tolba. "An Enhanced Test Case Generation Technique Based on Activity Diagrams". 2011 International Conference on Computer Engineering & Systems (ICCES). Nov. 29 2011.
- [56] Anthony Barrett, Daniel Dvorak, "A Combinatorial Test Suite Generator for Gray-Box Testing", Third IEEE International Conference on Space Mission Challenges for Information Technology. 2009.
- [57] Lingzhong Meng, Minyan Lu, Baiqiao Huang, Xiaojie Xu, "Using relative complexity measurement which from complex network method to allocate resources in complex software system's gray-box testing", International Symposium on Computer Science and Society, 2011.
- [58] Harrison Warren, Cook Curtis. "A micro/macro measure of software complexity[J]". Journal System Software. 1987 : 213-219

8. APPENDIX A: PROGRAMMING CODES

```
/*
 * Guest_A_Login.java
 *
 * Created on Mar 4, 2011, 12:22:38 AM
 */

package Hotel_mgt_Guest;

import javax.swing.JOptionPane;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.Arrays;
import javax.swing.JFrame;
import javax.swing.UIManager;

/**
 *
 * @author Mohammad Rava
 */
public class Guest_A_Login extends javax.swing.JFrame {

    // Data related to Database connection
    private Connection connect = null;
    private Statement statement = null;
    private ResultSet resultSet = null;

    String usr = null;
    char[] psw = null;

    String username;
    char[] password;
    String userPosition;
```

```

    public static int userID = 0;

    /** Creates new form Staff_A_Login */
    public Guest_A_Login() {
        initComponents();
    }

    @SuppressWarnings("unchecked")
    // <editor-fold defaultstate="collapsed" desc="Generated Code">
    private void initComponents() {

        jLabel1 = new javax.swing.JLabel();
        jLabel2 = new javax.swing.JLabel();
        jUserField = new javax.swing.JTextField();
        jPassField = new javax.swing.JPasswordField();
        btnLogin = new javax.swing.JButton();
        btnCancel = new javax.swing.JButton();

        setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);
        setTitle("Guest Login");
        setName("Form1"); // NOI18N
        setResizable(false);

        jLabel1.setText("Your ID:");

        jLabel2.setText("Password:");

        btnLogin.setText("Login");
        btnLogin.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                btnLoginActionPerformed(evt);
            }
        });

        btnCancel.setText("Cancel");
        btnCancel.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                btnCancelActionPerformed(evt);
            }
        });
    }

```



```

});

    javax.swing.GroupLayout layout = new
javax.swing.GroupLayout (getContentPane ());
    getContentPane ().setLayout (layout);
    layout.setHorizontalGroup(

layout.createParallelGroup (javax.swing.GroupLayout.Alignment.LEADING)
        .addGroup (javax.swing.GroupLayout.Alignment.TRAILING,
layout.createSequentialGroup (
            .addGap (33, 33, 33)

.addGroup (layout.createParallelGroup (javax.swing.GroupLayout.Alignment.LEADING)
            .addComponent (btnLogin,
javax.swing.GroupLayout.PREFERRED_SIZE, 70,
javax.swing.GroupLayout.PREFERRED_SIZE)

.addGroup (layout.createParallelGroup (javax.swing.GroupLayout.Alignment.TRAILING)
            .addComponent (jLabel2)
            .addComponent (jLabel1)))

.addPreferredGap (javax.swing.LayoutStyle.ComponentPlacement.RELATED,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)

.addGroup (layout.createParallelGroup (javax.swing.GroupLayout.Alignment.LEADING, false)
            .addComponent (jPasswordField)
            .addComponent (jTextField)
            .addGroup (layout.createSequentialGroup ()
                .addGap (59, 59, 59)
                .addComponent (btnCancel)))
        .addGap (31, 31, 31))
);

    layout.linkSize (javax.swing.SwingConstants.HORIZONTAL, new
java.awt.Component[] {btnCancel, btnLogin});

    layout.setVerticalGroup(

layout.createParallelGroup (javax.swing.GroupLayout.Alignment.LEADING)
        .addGroup (javax.swing.GroupLayout.Alignment.TRAILING,
layout.createSequentialGroup (
            .addContainerGap (24, Short.MAX_VALUE)

```

```

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)

        .addComponent(jLabel1)

        .addComponent(jTextField,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE))

        .addGap(18, 18, 18)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)

        .addComponent(jLabel2)

        .addComponent(jPasswordField,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE))

        .addGap(18, 18, 18)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)

        .addComponent(btnLogin)

        .addComponent(btnCancel))

        .addGap(25, 25, 25))

);

java.awt.Dimension screenSize =
java.awt.Toolkit.getDefaultToolkit().getScreenSize();

setBounds((screenSize.width-277)/2, (screenSize.height-182)/2, 277,
182);
}

private void btnCancelActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    JOptionPane.showMessageDialog(rootPane,
        "The Program is Terminating",
        "Exit Application",
        JOptionPane.INFORMATION_MESSAGE);

    System.exit(0);
}

private void btnLoginActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    usr = jTextField.getText();
    psw = jPasswordField.getPassword();

```

```

try {
    // This will load the MySQL driver, each DB has its own driver
    Class.forName("com.mysql.jdbc.Driver").newInstance();

    // Setup the connection with the DB
    connect = DriverManager
        .getConnection("jdbc:mysql://localhost:3306/hotelmgmt"
            , "root", "root");

    // Statements allow to issue SQL queries to the database
    statement = connect.createStatement();

    // Result set get the result of the SQL query
    resultSet = statement
        .executeQuery("select * from guest where Guest_ID= '" +
usr + "'");

    while (resultSet.next()) {
        userID = resultSet.getInt(1);
        username = resultSet.getString(1);
        password = resultSet.getString(12).toCharArray();

        if (usr.equals(username) && Arrays.equals(psw, password)){

            JOptionPane.showMessageDialog(rootPane,
                "Login was Successful\nWelcome to Hotel Management
System!",
                "Login Successful",
                JOptionPane.INFORMATION_MESSAGE);

            this.dispose();
            new Guest_Main_Menu().setVisible(true);

        } else {

```



```

        JOptionPane.showMessageDialog(rootPane,
        "Wrong Username and/or Password. \tPlease Try Again:",
        "Invalid Inputs",
        JOptionPane.WARNING_MESSAGE);
    }

}

catch (Exception e){
    String msg= e.getMessage();
    System.out.println(msg);
    e.printStackTrace();
}

}

/**
 * @param args the command line arguments
 */
public static void main(String args[]) {
    java.awt.EventQueue.invokeLater(new Runnable() {
        public void run() {
            try {
                UIManager.setLookAndFeel("com.sun.java.swing.plaf.nimbus.NimbusLookAndFeel");
            } catch (Exception ex) {
                ex.printStackTrace();
            }

            Guest_A_Login login = new Guest_A_Login();

            login.setUndecorated(true);

            login.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
            login.setVisible(true);

        }
    });
}

```

```
// Variables declaration - do not modify
private javax.swing.JButton btnCancel;
private javax.swing.JButton btnLogin;
private javax.swing.JLabel jLabel1;
private javax.swing.JLabel jLabel2;
private javax.swing.JPasswordField jPasswordField;
private javax.swing.JTextField jPasswordField;
// End of variables declaration
```

}



THE ASSUMPTION UNIVERSITY LIBRARY

