# TEST DATA GENERATION METHOD FOR DYNAMIC – STRUCTURAL TESTING IN AUTOMATIC PROGRAMMING ASSESSMENT

**MD. SHAHADATH SARKER**

**MASTER OF SCIENCE (INFORMATION TECHNOLOGY)**

**UNIVERSITI UTARA MALAYSIA**

**2016**

## Permission to Use

# Declaration

I declare that this thesis is my own work and has not previously been submitted in any form for another degree or diploma at any other university or other institute of tertiary education. Information derived from the published and unpublished works of others have been acknowledged in the text and a list of references is given.

Md. Shahadath sarker
2016

## Abstrak

Penaksiran Pengaturcaraan Automatik atau dikenali sebagai APA telah diketahui sebagai suatu keadah yang berkesan dalam membantu para pensyarah untuk melaksanakan penaksiran dan penggredan tugasan pengaturcaraan pelajar. Untuk melaksanakan pengujian dinamik dalam APA, adalah menjadi suatu keperluan untuk menyediakan  set data ujian melalui proses penjanaan data ujian yang bersistematik. Sekiranya memfokus kepada bidang pengujian perisian, pelbagai kaedah untuk mengautomasikan penjanaan data ujian telah dicadangkan.Walaubagaimanapun, kaedah-kaedah ini jarang diguna pakai di dalam kajian semasa APA. Terdapat kajian awalan yang cuba mengintegrasikan APA dan penjanaan data ujian, tetapi masih terdapat jurang dari segi menerbitkan dan menjana data ujian untuk pegujian dinamik-berstruktur. Untuk mengatasi jurang ini, kajian ini mencadangkan suatu kaedah penjanaan data ujian untuk melaksanakan pengujian dinamik-berstruktur (atau dikenali sebagai DyStruc-TDG).DyStruc-TDG direalisasikan sebagai alatan fizikal yang bertindak sebagai penjana data ujian untuk menyokong fungsian APA.Dapatan daripada ekperimen kawalan yang dilaksana berdasarkan reka bentuk *one-group pre-test* dan *post-test* mendapati bahawa DyStruc-TDG memperbaiki kriteria kecukupan data ujian kebolehpercayaan (atau pengujian positif) dalam penaksiran pengaturcaraan.Kaedah yang dicadangkan ini adalah dijangkakan dapat membantu para pensyarah kursus pengaturcaraan awalan untuk menerbitkan dan menjana data ujian dan kes ujian untuk melaksanakan penaksiran pengatucaraan automatik untuk pengujian dinamik-berstruktur tanpa memerlukan pengetahuan khusus dalam reka bentuk kes ujian.Dengan mengguna-pakai kaedah ini sebagai sebahagian APA, beban para penyarah secara tidak langsung dapat dikurangkan secara berkesan oleh kerana penaksiran tipikal yang manual senantiasa cenderung kepada ralat dan penyebab kepada ketidakseragaman.

**Kata kunci:** penjanaan data ujian, Penaksiran Pengaturcaraan Automatik, pengujian dinamik, pengujian berstruktur, path coverage, Modified Condition/Decision Coverage.

# Abstract

Automatic Programming Assessment or so-called APA has being known as a significant method in assisting lecturers to perform automated assessment and grading on students' programming assignments. Having to execute a dynamic testing in APA, it is necessary to prepare a set of test data through a systematic test data generation process. Particularly focusing on the software testing research area, various automated methods for test data generation have been proposed. However, they are rarely being utilized in recent studies of APA. There have been limited early attempts to integrate APA and test data generation, but there is still a lack of research in deriving and generating test data for dynamic structural testing. To bridge the gap this study proposes a method of test data generation for dynamic structural testing (or is called DyStruc-TDG). DyStruc-TDG is realized as a tangible deliverable that acts as a test data generator to support APA. The findings from conducted controlled experiment that is based on one-group pre-test and post-test design depict that DyStruc-TDG improves the criteria of reliability (or called positive testing) of test data adequacy in programming assessments. The proposed method is expectantly to assist the lecturers who teach introductory programming courses to derive and generate test data and test cases to perform automatic programming assessment regardless of having a particular knowledge of test cases design in conducting a structural testing. By utilizing this method as part of APA, the lecturers' workload can be reduced effectively since the typical manual assessments are always prone to errors and leading to inconsistency.

**Keywords:** test data generation, Automatic Programming Assessment, dynamic testing, structural testing, path coverage, Modified Condition/Decision Coverage.

# Acknowledgement

Firstly, it is my great pleasure to thank Allah (swt) that has given me opportunities to complete this thesis. A billion thanks those, who directly or indirectly contributed in the development of this work and who influenced my thinking, behavior, and acts throughout the study period.

I express my sincere gratitude to my supervisor Dr.Rohaida Binti Romli for her continuous support, cooperation, valuable suggestions and motivation for completing the thesis and she exchanged her interesting ideas, thoughts and made this thesis easy and accurate.

Lastly, I would like to thank all of my friends for their encouragement and valuable suggestions during my study period.

# Table of Contents

## List of Tables

## List of Figures

# List of Abbreviations

| | |
|---|---|
| APA | Automatic Programming Assessment |
| ATDG | Automatic Test Data Generation |
| DyStruc-TDG | Dynamic Structural- Test Data Generation |
| MC/DC | Modified Condition/Decision Coverage |
| OOAD | Object Oriented Analysis and Design |
| UML | Unified Modeling Language |
| UUM | Universiti Utara Malaysia |

# CHAPTER 1

# INTRODUCTION

## 1.1 Background of Study

Learning computer programming languages has become essential for students who pursue their study in Information Technology, Computer Science and Software Engineering disciplines. Computer introductory programming courses are commonly offered for the first year degree students who pursue their study in these fields (Truong *et al.*, 2005). Effective and good programming skills are necessary for students in order to be a master in programming. Students can be skilled in programming only through practices (Lahtinen *et al.,* 2005). Computer programming courses are normally designed with full of practical besides theory. The goal of practical course is to develop student's basic understanding of programming principles and writing basic source code. Therefore, students are given many programming exercises as take home assignments or hands on practice in the class in order to develop student's programming understanding and skill (Rohaida *et al*., 2010). If the assessment of programming exercises is done by manually for a large number of students it leads to workload to lecturers and assessing manually is really difficult task which cannot ensure the consistency and accuracy of the marking scheme (Rohaida *et al*., 2010). Therefore, the concept of Automatic Programming Assessment (APA) has become very important to assess students program for grading and providing feedback (Saikkonen *et al*., 2001). Besides, APA can improve students marking assessment in terms of consistency and thoroughness testing (Gupta *et al*., 2012).

According to Jackson (1996) APA is founded on software testing technique. The programming assessment normally involves the measuring of the program quality. In order to achieve program quality the program should be tested. Hence, through the software testing technique the quality of the program can be measured (Rohaida *et al*., 2010). Software testing is a method for locating, measuring, and disclosing errors that occurred in a program (Latiu, 2012). Software testing can be categorized into two types: static analysis and dynamic testing, in which static testing is a test that does not involve in the execution of the program (Zin *et al*., 1994). On the other hand, dynamic testing requires a program execution with test data (Chu *et al.,* 1997). Test data is data which is developed as input in order to perform testing for any software program (Korel, 1990).

Furthermore, dynamic software testing can be classified into two main categories: black-box testing and white-box testing (Sommerville, 2001). According to Gentiana (2012), black box testing technique focuses program outputs and it does not need to access the source code of the program. Besides, white box is a testing technique that refers to the structure of the internal program (Varshney *et al.*, 2013). Thus, in order to perform white box testing or so called structural testing it is required to generate test data.

Test data generation in program testing refers to the process of identifying a set of test data, which satisfies given testing criterion (Korel, 1990). On the other hand, it is also important to select appropriate test data to cover good test coverage. Designing test data that is adequate in testing is important to ensure that testing is done thoroughly enough. The main reason of selecting appropriate test data is to overcome ambiguous feedbacks because it might interpret wrong result for students programming assessment (Jackson, 2000). Generation of test data manually is very difficult and time consuming. It is also costly, creates error, and incomprehensive (Latiu *et al*., 2012). It is very hard and required much effort from human in order to continue the process of generating test data smoothly (Monpratarnchai *et al.,* 2014).

To date there have been limited studies attempted to integrate Automatic Test Data Generation (ATDG) with APA (Malmi *et al.,* 2004; Shukur *et al.,* 2005;Ihantola, 2006; Tillman *et al.,* 2013; Hakulinen and Malmi, 2014). Their detailed studies will be further discussed in **Chapter 2**. Based on previous study, there is a limitation of adopting structural code coverage in test data generation for APA. Therefore, the attempt of generating automatic test data for structural code coverage with integration in APA is still in preliminary stage.

## 1.2 Problem Statement

The goal of computer programming courses is to increase student's knowledge of solving programming problems and understanding the principles. This course is more practical that is why students are given more exercises to practice. Due to a large number of students in the class, it's very hard to assess the programming exercises manually by the lecturers (Rohaida *et al.,* 2010). Manually assessing the students program it takes much effort and time consuming (Jackson, 1996). It also may allow unintended biases and different standard of marking schemes

(Rohaida *et al.,* 2010). Therefore, automatic programming assessment plays an important role regarding assessing students programming exercises and providing feedback to them (Saikkonen *et al*., 2001). Thus, there are several tools were developed to assess students programming exercises automatically such as Assyst (Jackson and Usher, 1997), BOSS (Lucky and Joy, 1999), TRAKLA2 (Malmi *et al.,* 2004), PASS (Choy *et al.,* 2005) and others. Unfortunately, most of these automated tools do not provide the means of generating test data automatically.

APA involves a set of test data to execute dynamic testing on students program (Rohaida *et al.,* 2010). Manually test data generation is labour demanding, expensive and prone to errors. Therefore, it is required to generate test data automatically for APA. To date, there are various automated test data generations methods are available in industry (Clarke *et al*., 1976; Gupta *et al*., 1998; Offutt *et al*., 2003; Zamli *et al*., 2007; Zidoune *et al*., 2012; Pargas *et al*., 1999). Unfortunately, it appears in the recent studies on APA very few of them utilized these methods (as stated in Section 1.1). Some studies such as Guo *et al.* (2010) and Cheng *et al.* (2011) have used external automated test data generation tool which is either a product of past research within the same institution or a commercialized product that incurs cost. Thus far, merely limited studies (Ihantola, 2006; Tillman *et al*., 2013; Rohaida, 2014) that have attempted to integrate APA and Automatic Test Data Generation (ATDG) for structural testing. Their detailed studies will be further discussed in **Chapter 2**. In structural testing, path coverage criterion is most popular in terms of generating test data for programming assessment. In order to cover more thoroughness of testing and generating adequate test data, MC/DC coverage criterion is integrated with path coverage.

In order to design test set or test data in a structural testing it is important to ensure that testing is done thoroughly enough. This thoroughness will determine how adequate the testing is (Hayhurst, 2001). Thus, adequate testing depends on test adequacy criteria. According to Zhu *et al.,* (1997) test data adequacy criteria can be categorized into two types: Specification based and Program based. In program based which specifies the testing of the program under test and decides if a test set is adequate according to whether the program has been thoroughly exercised. The test adequacy criteria are very important because it distinguishes good test cases from bad ones and determines whether the testing is sufficient or not (Zhu, 1995). Structural testing can be described based on concept control flow graph model. This control flow graph model focuses on code coverage in structural testing. The most common control flow coverage criteria are path coverage, statement coverage, branch coverage, multiple condition coverage, condition/decision coverage and Modified Condition/Decision Coverage (MC/DC) coverage (Kamran *et al.,* 2009).

Some of the studies seemed adopting simple techniques to generate test data without focus on its internal logic and its coverage such as JTst (Zamli *et al*., 2007). Besides, the studies have not sufficiently included the ideal test criterion to derive adequate test data. Addressing these issues, this study attempts to generate automatic test data for dynamic structural testing in APA that do satisfy the test adequacy criteria for Path coverage and Modified Condition/Decision Coverage (MC/DC).

## 1.3 Research Questions

The main research question of this study is formulated as "***How to generate a set of test data that do satisfy the test adequacy criteria to adhere the coverage of structural testing of a program executed for APA***".

Based on the above main research question the following specific sub questions are identified:

1. How to construct a test data generation method to derive an adequate set of the test data for dynamic structural testing in APA?(RQ-1)

2. How to measure the adequacy of test data as derived in (1) in the context of APA? (RQ-2)

## 1.4 Research Objectives

The main goal of this study is to propose a test data generation method to perform a dynamic structural testing for APA. This method is called DyStruc-TDG.

In order to achieve the main goal the following objectives are formulated:

1. To construct a test set which include an adequate of test data to represent DyStruc-TDG (RO-1)

2. To measure the adequacy of test data derived from DyStruc-TDG in the context of programming assessment(RO-2)

## 1.5 Scope of the Study

Automatic assessment and software testing are definitely involved broad topics. The scope of the study includes:

i) Test Data Generation for APA

The study focuses only from the aspect of programming assessments in academic rather than in the software testing industry. Programming assessment is considered as essential elements of software engineering and computer science education. The major aim of this study is to generating and deriving a set of test data which will be used for testing student's program in APA. The assessment will involve the process of judging certain software quality factors included in each students programming solution. This particularly has a direct influence to a concept of test adequacy criteria.

ii) Dynamic Structural Testing

Software testing is generally categorized as static and dynamic testing. Dynamic testing can be divided into another two parts that are functional and structural testing respectively. Functional testing is known as black box testing and, conversely, structural testing is known as white box or glass box testing. This study focuses only dynamic structural testing and the criteria of reliability (or so-called positive testing) of test data adequacy. The criterion of positive testing is with regard *"the program does what it is supposed to do"* that is basically concerned with what operations or/and conditions that a program intends to accomplish and to satisfy (IPL, 1997a; Walkins and Mills, 2011). An integration of two prominent structural code coverage that are path coverage and Modified Condition/Decision Coverage (MC/DC) has been proposed in this study to derive and generate the required test data and test set for dynamic-structural testing.

iii) Quality factor of correctness

In software testing, there are many software quality factors are identified. Based on literature review, most of the studies in APA focus on the quality of correctness rather other software quality factors. Correctness is concerned with a quality attribute that can be defined as the degree to which the program performs it intended functions (Burnstein, 2003). Thus, this study focuses on this software quality factor.

iv) Introductory programming courses

Introductory programming courses are common for the first year degree students in IT or computer science related area. Programming courses are designed with more practical rather than theory. As a result students are given many exercises in order to excel in programming courses. As nowadays Java is most common programming languages are taught in the universities, hence this study focuses only the introductory programming course that utilises java as the programming language.

## 1.6 Contribution of the Study

This study is an attempt to adapt existing test adequacy criteria applied in software testing field as a test data generation method to perform a dynamic structural testing in APA. This study contributes in terms of theoretical and practical context. In theoretical perspective, this study enhances the existing researches in APA by providing the means of deriving and generating test data automatically by integrating the path and MC/DC coverage. These two structural codes coverage provide a significant impact in reducing the number of test cases required to test students' programming solutions in terms of the aspect of structural testing. In addition, a more thorough testing aspect is considered as each individual condition has been a part of the means of deriving the test cases.

On the other hand, in practical perspective, this study contributes a physical deliverable that is a test data generator to provide a medium of generating test data automatically for the usage of lecturers. Though this generator, it is able to assist the lecturers who teach programming courses to generate test data and test cases to perform automatic programming assessment regardless of having a particular knowledge of test cases design. Besides, indirectly the lecturers' workload can be reduced effectively since the typical manual assessments are always prone to errors.

## 1.7 Organization of the Thesis

This thesis is comprised of six chapters, namely introduction, literature review, methodology, proposed work of DyStruc-TDG, evaluation and conclusion. The brief explanation of each chapter is as follows:

**Chapter 1** is the introductory part of this thesis. In this chapter it includes background of study, problem statement, research questions, research objectives, scope of the study and finally contribution of the study.

**Chapter 2** reviews the existing work of this study. Mainly, most related contents are reviewed in this chapter, namely software testing, APA, ATDG and integration of APA with ATDG. The discussion has focused more on related work of automatic programming assessment and its integration with test data generation.

**Chapter 3** is about research methodology. This chapter explains in details the research procedure which is carried out throughout the study. In summary, there are four phases included: theoretical study, construction of method, development of prototype and finally evaluation and conclusion. Each phase has its own activities and outcomes.

**Chapter 4** provides the explanation of proposed work for the research in this study. The proposed method is called DyStruc-TDG which includes an adequate set of test data to perform dynamic structural testing of a program for APA. DyStruc-TDG covers both control structures which are selection control structures and repetition control structures. The selected two test adequacy criteria which are path coverage and Modified condition/Decision coverage (MC/DC) also explained for both control structures.

**Chapter 5** explains the evaluation process and analysis of the obtained results. The evaluation process involves controlled experiment with pre-test and post-test questions and comparative evaluation.

**Chapter 6** is the last chapter that concludes the study. It includes the conclusion of accomplishment each research objectives as well as future work.

# CHAPTER 2
# LITERATURE REVIEW

## 2.1 Introduction to Software Testing

Software testing is an important technique in measuring the quality of software product assurance (Wegener, 2001). Software testing is a continuous procedure where it compares the actual result with expected result (Latiu, 2012). Software testing is a set of actions applied with the aim of getting errors and also make sure that the system is running according to the provided specification (Sapna, 2013). There are two aims of software testing such as fulfilling customer requirements and reveal the bugs (Khan, 2011). Software testing is essential to ensure software quality. Software testing is very expensive and it takes a lot of time to conduct (Tracey *et al.*, 1998). An important aspect of software testing is being able to decide when enough testing has been performed (Zhu *et al.*, 1993). The following Figure 2.1 shows the types of different software testing techniques.

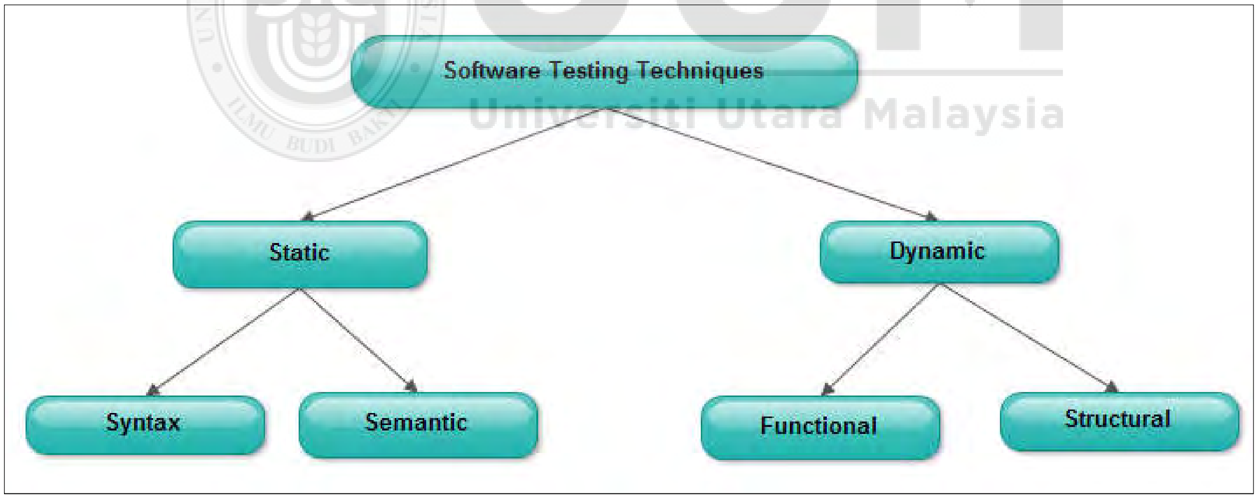


Figure 2.1 Types of software testing techniques

In software testing, static testing is done without execution of program. Static testing involves manual or automated reviews of the project documents, software models and codes (Bertolino *et al.*, 2005). On the other hand, dynamic testing is one kind of technique where it is required to execute the code in order to perform testing (Gupta *et al.*, 2012). Thus, it is required to generate test data to test the program. In this study, test data will be generated based on dynamic structural testing. The next section will discuss about dynamic structural testing.

## 2.2 Dynamic Structural Testing

By default, test data generation plays an important role in software testing, particularly to perform a dynamic testing. Test data generation techniques of dynamic structural are relied on the execution. According to Korel (1990) these procedures might be classified into several types such as random test data generation, structural or path-oriented test data generation, gold oriented test data generation and data specifications test data generation.

Structural testing (or white-box testing) is a method of testing the internal structure of software and programming applications and also known as clear box testing, glass box testing, transparent box testing and structural testing (Latiu, 2012). In structural testing, the internal structure and its behavior are inspected by executing the code.

In order to effective dynamic program analysis, it is required to execute the targeted program with enough test data's. Thus, an adequate set of test data will ensure how adequate the testing is (Gupta *et al*., 2012). The next section will elaborate about test adequacy criteria.

## 2.3 Test Adequacy Criteria

According to Zhu (1995) the central problem of software testing is "What is test data adequacy criterion?" Software test data adequacy criteria are rules that need to determine whether the software has been tested sufficiently or not (Zhu, 1995). Good test cases and bad test cases can be distinguished based on test adequacy criterion and can be determined when to finish the testing process (Sapna, 2013). There are mainly two groups of adequacy criteria based on program based structural test adequacy: Control Flow Adequacy and Data Flow Adequacy (Zhu *et al*., 1997). However in this study only control flow adequacy criteria will be discussed because the control flow graph model focuses on code coverage in structural testing. The most common control flow coverage criteria are path coverage, statement coverage, branch coverage, multiple condition coverage, condition/decision coverage and MC/DC coverage (Kamran *et al.,* 2009). A flow graph model is a directed graph which consists of nodes and edges (Zhu *et al*., 1997). Nodes represent a linear sequence of computations and edges represents transfer of control in an ordered pair of nodes. In order to know how adequate the testing is needs to test the coverage of the program. Coverage is a measure and it refers to the extent to which a given verification activity has satisfied its objectives (Hayhurst, 2001).

Goodenough and Gerhart's (1975) pointed out that a software test adequacy criterion is a predicate that defines "what properties of a program must be exercised to constitute a 'thorough' test, that is, one whose successful execution implies no errors in a tested program." Thus, in order to guarantee the correctness of adequately tested programs, they proposed *reliability* and *validity* requirements of test criteria. Reliability requires that a test criterion always produce consistent test results; where validity requires that the test always produce a meaningful result. In this study, only reliability test adequacy criteria will be incorporated in deriving test data to cover structural testing. However, in this study the terms of positive testing is also used as alternative term for reliability test adequacy criteria.

Kamran (2009) reported the most common control flow coverage criteria are statement coverage, path coverage, branch coverage, condition coverage, multiple condition coverage, modified condition/decision coverage (MC/DC) and loop coverage. Based on the survey conducted the ranking of importance code coverage for programming assessment is shown in Figure 2.2(Abdurahim, 2014)

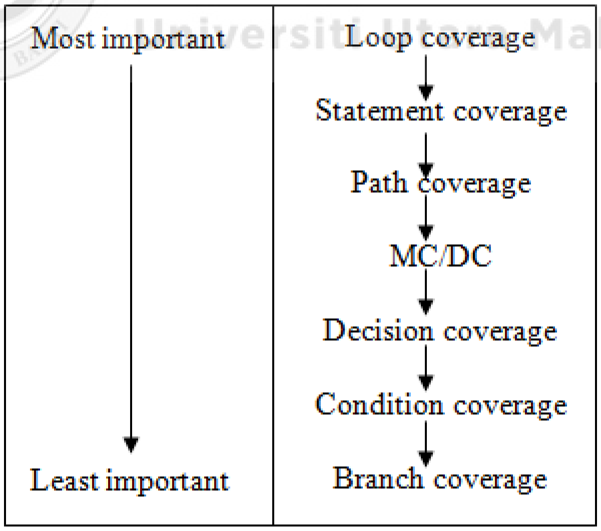


Figure 2.2 Ranking of code coverage based on survey (Abdurahim, 2014)

The following section will describe about all code coverage metrics as well as the comparison of the most applied coverage metrics in APA in order to adopt coverage metric for this study.

### 2.3.1 Loop Coverage

Loop coverage explains whether each loop counter flow program will zero again, once just once or more than once in a row. One of the more useful suggests that a loop is covered if in at least one test the body was executed exactly once, and if some test the body was executed more than once. The loop path selection is one of the important processes and its result will directly affect the accuracy of one path and then leads to the inaccurate generation of test cases (Qiang, 2013). The following Figure 3.2 shows an example of loop coverage code segment.

In Java programming there are three types of loop statements: *for* loops, *while* loops and *do-while* loops (Liang, 2009). The *while loops* and *do-while* loops are quite similar with each other except execution behavior. The *"for loop"* is involved with counter value. The details explanation of these three types of loops coverage is provided below.

#### *(i) for loop*

The following Figure 3.2 shows the code segment of "*for loop*" coverage in a program.

```
for(inti=1; i>2; i=i+1){ [S1]
            System.out.println("Welcome"); [S2]
                        System.out.println("\n"); [S3]
}
```
Figure 2.3 Code segment for "for loop" control structures

Figure 2.4 describes the flow graph of the above code segment for "for loop" coverage.

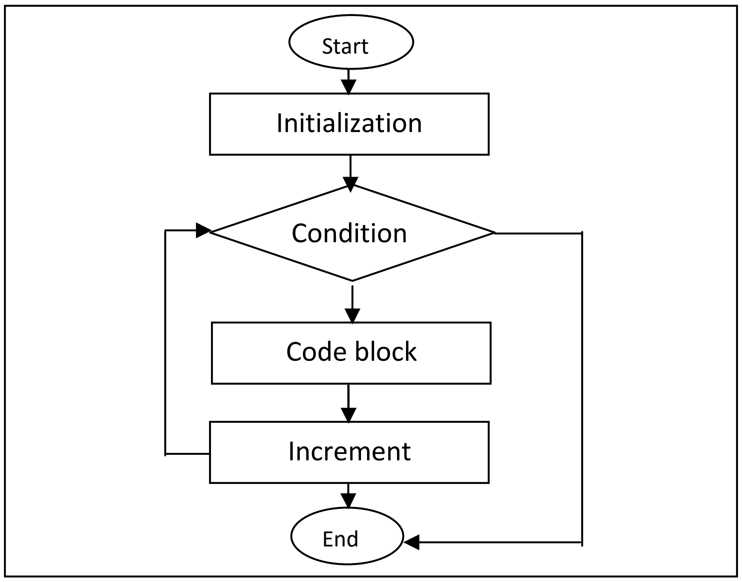


Figure 2.4 Flow graph of code segment for loop control structures

Based on the above Figure 2.3 the following Table 2.1 shows the number of test cases, sequence of executed statements and times of loop execution.

Table 2.1 Test cases for loop coverage

| Test Cases (TC) | Sequence of executed statement(s) | Loop execution |
|---|---|---|
| TC1 | S1 | Zero |
| TC2 | S1,S2,S3,S1 | Once |

**(ii)** ***while loop***

The following Figure 2.5 shows the code segment of "*while loop*" coverage in a program.

```
public void WhileLoop(){
        int x=5;
        while(x>0){ [S1]
                        System.out.println("Welcome"); [S2]
                                        System.out.println("\n");[S3]
        }
        System.out.println("fail");[S4]
}
```

Figure 2.5 Code segment for while loop coverage

Figure 2.6 describes the flow graph of the above code segment for "*while loop*" coverage

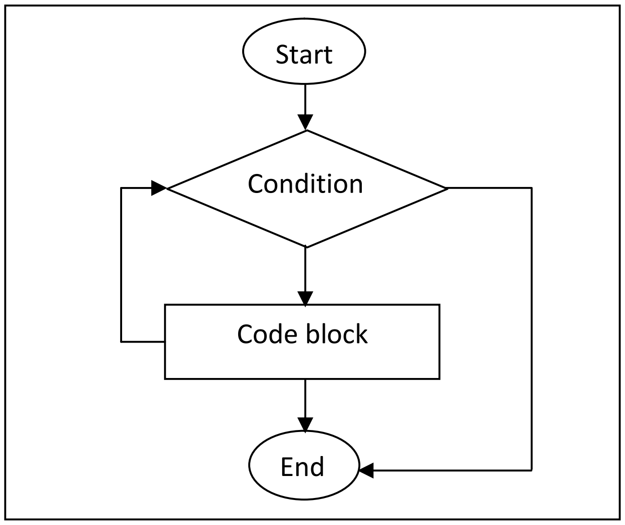


Figure 2.6 Flow graph of while loop coverage

Based on the above Figure 2.6, the following Table 3.3 shows the number of test cases, test data, number of times loop execution and executed statements.

Table 2.2 Test cases for while loop coverage

| Test Cases | Test Data | Loop execution | Statements Execution |
|---|---|---|---|
| TC1 | 5 | Once | S1, S2,S3 |
| TC2 | 0 | Zero | S1, S4 |

**(iii)** ***do… while*** **loop**

The following Figure 2.7 shows the code segment of ***do…while*** loop coverage in a program.

```
public void WhileLoop(){
      int x=1;
      do{[S1]
                System.out.println("value:"+x); [S2]
                           X++; [S3]

      System.out.println("\n"); [S4]
      }
      while(x<=5);[S5]
}
```

Figure 2.7 Code segment for statement coverage

Figure 2.8 describes the flow graph of the above code segment for *do…while* loop coverage

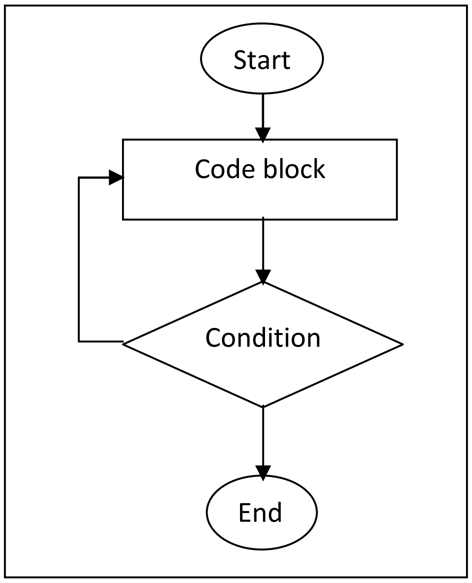


Figure 2.8 Flow graph of *do…while* loop

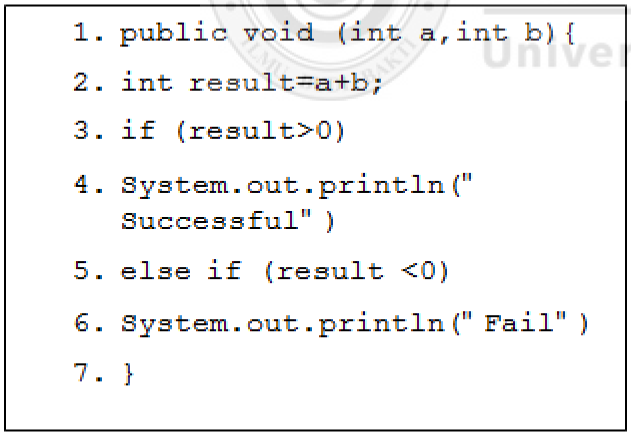
Based on the above Figure 2.8, the following Table 2.3 shows the number of test cases, test data, number of times loop execution and executed statements.

Table 2.3 Test cases for do…while Coverage

| Test Cases | Test Data | Loop execution | Statements Execution |
|---|---|---|---|
| TC1 | 1 | 5 times | S1, S2, S3, S4, S5 |
| TC2 | 6 | once | S1, S2, S3, S4, S5 |

### 2.3.2 Statement Coverage

Statement coverage is one example of control flow based adequacy criteria and it needs each statement in the program to go through and be implemented at least once (Zhu, 1995).Statement coverage also called line coverage for structural testing.  According to Myers (1979)"statement-coverage criterion is so weak that it is generally considered useless." After execution it is compared to a list of all execuTable statements. According to Joseph (2005) in statement coverage all the test cases need to perform in a program. The following Figure 2.9 is one example of code segment of statement coverage.



```
1. public void (int a,int b){
2. int result=a+b;
3. if (result>0)
4. System.out.println("
   Successful")
5. else if (result <0)
6. System.out.println(" Fail")
7. }
```

Figure 2.9 Code segment for statement coverage

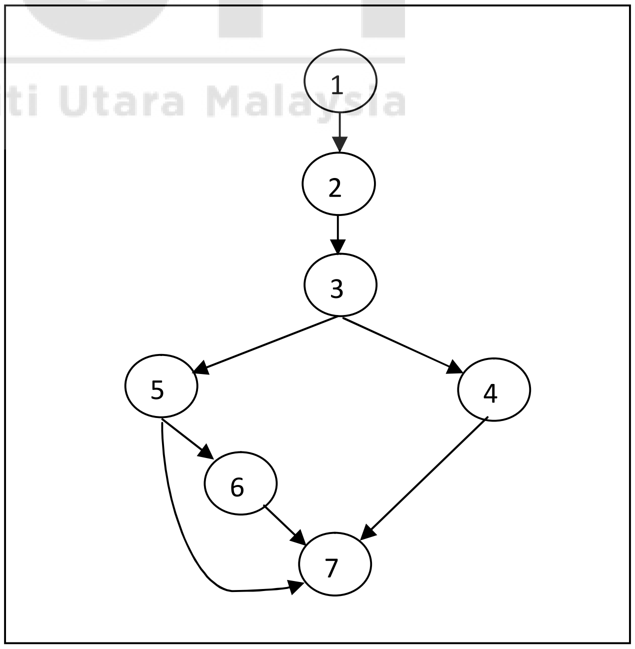


Figure 2.10 Flow graph of statement coverage

Figure 2.10 shows the flow graph of statement coverage for the code segment of Figure 2.9. Based on the code segment of Figure 2.9, the following Table 2.4 describes the number of test cases, test data and the statement coverage.

Table 2.4 Test cases for Statement Coverage

| **Test Cases (TC)** | **Test Data** | | **Statement Coverage** |
|---|---|---|---|
| | **a** | **b** | |
| TC 1 | 3 | 8 | 1, 2, 3, 4, 7 |
| TC 2 | -5 | -6 | 1, 2, 3, 5, 6, 7 |
| TC 3 | 0 | 0 | 1, 2, 3, 5, 7 |

### 2.3.3 Path Coverage

This testing is a kind of structural testing in which the source codes of the program to locate each way possible each program through which it passes (Latiu, 2012). It is also one of the ways to test the route to make sure it passes all the way at least once in the program. Furthermore, path testing is also found in control flow structure. The method used to produce the test of the control program is to select routes and how to determine the values of the input. The Figure 2.11 is an example of control flow graph which shows possible paths from start node to end note.

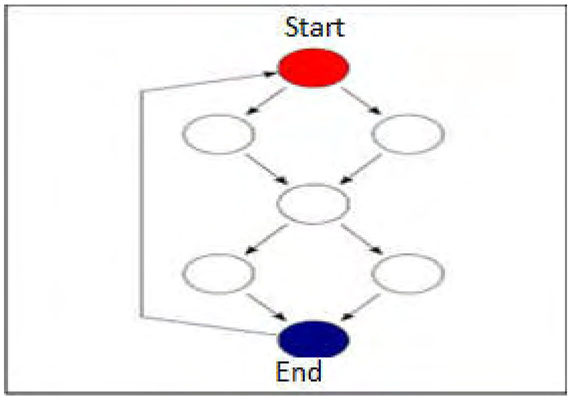


Figure 2.11 Control flow graph

When the set of selected the path regularly and properly, this indicates that the access of choosing the path is thoroughness (Beizer, 1990). The objective of the path testing is to ensure that every path in the program travelled through programs executed at least once. In structural test data generation, test cases are derived in such a way that every path is executed at least once as path coverage. It ensures coverage of all the paths from start to end.

In order to derive test data for structural test data generation it relies on mainly two control structures (Malik and Burton, 2009; Lewis *et al*., 2008) which are:

1. Selection or conditional statement, it is also called a branch
   - The program executes particular statements depending on one or more conditions
2. A loop or repetition statement
   - The program repeats particular statements certain number of times depending on one or more conditions.

These two control structures consist of Boolean/logical expression, which is also called condition. Each condition in the expression evaluates either true or false (Levis *et al*., 2008).

Based on Java programming, selection or conditional statements concern *if, if…else* and *switch…case* decision making statements (Liang, 2009). The following section will describe path coverage in details for Selection control structures and Loop control structures.

**(i) Path coverage (Selection control structures)**

The deriving of test cases of this type of decision making structure relies on the following properties:

1. The number of selection control statement
2. The number of decisions of each selection statement
3. The category of input conditions (valid and invalid)

Example:

```
if (average>=90) && (average<=100){
        System.out.println(" Excellent");
}
else{
        System.out.println(" Fail, try again");
}
```

Figure 2.12 Code segment for selection control structures

Table 2.5 Test cases for path coverage

| **Test Case** | **(average>=90)<br>E1** | **(average<=100)<br>E2** | **E1&&E2** |
|---|---|---|---|
| TC1 | T | T | T |
| TC2 | T | F | F |
| TC3 | F | T | F |
| TC4 | F | F | F |

**(ii) Path coverage (Loop control structures)**

In Java programming there are three types of loop statements: *while* loops, *for* loops and *do-while* loops (Liang, 2009). The *while* loops and *do…while* loops are quite similar with each other. The *for* loop is involved with counter value. The **Section 2.3.1** (loop coverage) has described more about loop control structures.

The next section will describe about Modified Condition/Decision Coverage (MC/DC) test adequacy coverage criteria.

### 2.3.4 Modified Condition/Decision Coverage (MC/DC)

The MC/DC is one of the structural coverage criteria that is used to assist in the assessment of adequacy and also one of the criteria's to explain that every condition separately change the result (Hayhurst, 2001). The decision has taken all possible outcomes at least once and we also say we cover both the true and the false branch. Every condition in then decision independently affects the decision's outcome. MC/DC is more practical criterion and typically a testing requirement for vital systems which developed in the avionics field (Kamran, 2009). The following section will describe the Modified Condition/Decision Coverage (MC/DC) in details for Selection control structures and Loop control structures.

**(i) Selection control structures (Modified Condition/Decision Coverage)**

MC/DC coverage criteria are used in order to reduce the number of test cases and time. At the same time, it makes sure the coverage of all statements. In order to derive test cases, MC/DC depends on the following properties:

1. Every point of entry and exit in the program has been invoked at least once.
2. Every decision in the program has taken all possible outcomes at least once.
3. Every condition in a decision in the program has taken all possible outcomes at least once.
4. Every condition in a decision has been shown to independently affect that decisions outcome.

Example:

```
if ((x&&y)||z){
      System.out.println(" True" );
}
else{
      System.out.println(" False" );
}
```

Figure 2.13 Code segment for selection control structures (MC/DC)

Table 2.6 Test cases for MC/DC coverage

| Test Case | X | Y | Z | X&&Y =A | A\|\|Z |
|---|---|---|---|---|---|
| TC1 | T | T | F | T | T |
| TC2 | F | T | F | F | F |
| TC1 and TC2 show independence of X (covers X) | | | | | |
| TC3 | T | T | F | T | T |
| TC4 | T | F | F | F | F |
| TC3 and TC4 show independence of Y (covers Y) | | | | | |
| TC5 | F | F | T | F | T |
| TC6 | F | F | F | F | F |
| TC5 and TC6 show independence of Z (covers Z) | | | | | |

**(ii) Loop control structures (Modified Condition/Decision Coverage)**

According to Liang (2009) Java programming consists of three loops namely *for* loops, *while loops* and *do-while* loops. The while loops are similar with do while loops. It is different only in terms of execution. Besides, these two loops there is another loop called *for* loop. *For* loop is used as a counter. In general, it is compulsory for each loop in a program to have a valid input condition of the loop as one test case.

The MC/DC coverage criterion for Loop control structures also follows the same properties as described in **Section** 2.3.4 (i) for selection control structures in terms of MC/DC coverage.

The important aspect of this criterion is the requirement that testing should demonstrate the independent effect of atomic boolean conditions on the boolean expressions in which they occur (Rayadurgam *et al*., 2003). Based on details explanation of MC/DC coverage criterion concept in **Chapter 4** the following Table 2.7 concludes the formula for generating test cases by applying MC/DC coverage criterion.

Table 2.7 Formula obtained from MC/DC coverage

| No of Options (N) | Truth Table ($N^2$) | MC/DC (N+1) |
|---|---|---|
| 1 | 1 | 2 |
| 2 | 4 | 3 |
| 3 | 9 | 4 |
| 4 | 16 | 5 |
| 5 | 32 | 6 |

### 2.3.5 Multiple Condition Coverage

Multiple condition coverage ensures that each possible combination of outcomes to a decision is tested at least once (Hayhurst, 2001).Each possible outcome consists of only two values which are true and false. In order to perform structural testing for multiple condition coverage it is required $\mathbf{2^n}$ test cases where **n** is number of conditions in a decision.

### 2.3.6 Condition Coverage

This type of coverage is also one technique in software testing family. It is based on the logical flow of control within a program (Jefferson, 1996). This condition coverage requires that the assessment made available conditions. It goes through the path and stops on the conditional and condition in which decisions are taken at least once, but not all decisions will be taken. Besides that, condition coverage will include two criteria's where each entry and exit points will be conducted at least once (Hayhurst, 2001).

### 2.3.7 Branch Coverage

Branch coverage requires all branches and a decision which must be taken in the program. All paths branch to be passed at least once (Zhu, 1995). Branch coverage achieved when every path from a node is performed minimum one time. Example of branch coverage is if at least one true and one false evaluation for each predicate. It is also widely used because of its ease of implementation and it is low overhead on the execution of the program (Wei, 2012).

The above **Section** (2.3.1 to 2.3.7) has described about the code coverage criteria for structural testing. Based on ranking of code coverage (Abdrahim, 2014) it shows the ranking of code coverage from most important to least important in programming assessment. The most popular coverage criterion is path coverage. This path coverage also includes all loop paths and statements in a program. Thus, it is a strong coverage criterion in programming assessment. On the other hand, MC/DC coverage criterion considers each condition individually in Boolean expression which provides more thoroughness of testing and generates adequate test cases.

## 2.4 Comparison of Different Code Coverage

According to Hayhurst *et al*., (2001) the following Table 2.8 shows the differences of code coverage based on their coverage criteria.

Table 2.8 Differences of code coverage based on coverage criteria (Hayhurst *et al*., 2001)

| Coverage Criteria | Statement coverage | Decision Coverage | Condition Coverage | Condition/ Decision Coverage | MC/DC | Multiple Condition Coverage | Path Coverage |
|---|---|---|---|---|---|---|---|
| Every point of entry and exit in the program has been invoked at least once | | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Every statement in the program has been invoked at least once | ✓ | | | | | | ✓ |
| Every decision in the program has taken all possible outcomes at least once | | ✓ | | ✓ | ✓ | ✓ | ✓ |
| Every condition in a decision in the program has taken all possible outcomes at least once | | | ✓ | ✓ | ✓ | ✓ | ✓ |
| Every condition in a decision has been shown to independently affect that decision's outcome | | | | | ✓ | ✓ | |
| Every combination of condition outcomes within a decision has been invoked at least once | | | | | | ✓ | ✓ |

Based on the discussion, above Table 2.8 shows the differences of code coverage based on their coverage criteria. As mentioned earlier that the thoroughness of testing determines adequate of testing. According to Goodenough and Gerhart (1975) the term “thoroughness” testing refers to successful execution without error in a tested program. In order to guarantee the correctness of adequately tested program, they proposed two requirements of testing criteria: *Reliability* and *Validity* test adequacy criteria. In this study, the *reliability* test adequacy criteria have been focused in order to derive test data for structural testing.

From the Table 2.8 it shows that Path coverage and MC/DC coverage criteria fulfill the maximum coverage of testing. Thus, the Path Coverage and Modified Condition/Decision Coverage (MC/DC) has been adopted in this study. The next section will describe about automatic programming assessment.

## 2.5 Automatic Programming Assessment (APA)

Automatic Programming Assessment (APA) is a great attraction in education domain. APA can decrease the burden of teaching instructor and time consuming and it aims to overcome the manual assessment. It is more convenient for students because they can get results as soon as possible when they submit their assignment or exercises to APA system. Feedback from the automatic programming assessment can help students learning process faster by getting their result instant. In the research there are few automatic programming assessment systems give critical, rich and timely feedback (Liang, 2009).

Furthermore, the instant formative feedback along with the results can significantly enhance students learning. A few automated programming assessment systems provide critical, rich and timely feedback which is of great enhancement for students in learning programming (Liang, 2009). The generation of test data process is an important part to carry out programming assessment of students. Dynamic testing was focused in the recent studies comparing static analysis for automated programming assessment (Jackson, 1996; Syukur, 1999; Shaffer, 2005). There are number of automatic tools are developed to assist APA. Such as Assyst, BOSS, GAME, TRAKLA2, PASS, ELP and others. (Jackson *et al.,* 1997; Schreiber, 1995; Korhonen, 2003).

According to literature survey (Rohaida, 2014) of APA research trends, the following Table 2.9 summarizes the trends of APA.

Table 2.9 Summary of the trends of APA (Rohaida, 2014)

| Criteria | **Testing Method** | **Quality Factor** | **Testing Category** | **TDG Technique** |
|---|---|---|---|---|
| | Dynamic Testing | Correctness | Black-box Testing | Manually |
| Description | Based on literature survey, most of the studies have performed dynamic testing instead of static analysis in APA. Static analysis mainly focuses on syntax analysis where dynamic testing concentrates on test coverage. Hence, Dynamic testing is used in APA. | Correctness is the most popular quality factor in testing because it provides satisfaction to the lectures in terms of assessing students program. | Black box testing is used extensively in APA rather than white box testing.<br><br>Black box testing also called as functional testing which identifies the correctness factor of a program. | In terms of assessing students programming exercises most of the studies derived test data manually. In some studies proposed JUnit testing framework which is required extensive technical skill. However, in order to generate automated test data two studies have utilized external tool (Goa et al., 2010; Cheng et al., 2011). Due to cost and accessibility it is difficult to utilize external tools to generate test data. |

In conclusion, the purpose of developing APA is to provide the facilities to the lecturers in term of reducing workload and time for assessing students programming assessment. As mentioned in problem statement that there is a need to generate automatic test data for structural testing. The next section will discuss about automatic test data generation.

## 2.6 Automatic Test Data Generation (ATDG)

Researches on automation of test data generation are commonly based on structural, functional test data generation techniques. Automation of structural testing is based on either static or dynamic testing criteria, and it has been widely investigated subject among researches. According to Gentiana *et al.*, (2012) test data pass through the intentional path for a given problem which follows five steps. The five steps are construction of a control flow graph, selection function construction, fitness function, program instrumentation, test data generation and execution of instrumented program. In order to overcome path testing problem there are different approaches of automated test data generation technique.

Edvardsson (1999) presented a test data generator system which comprises of three parts such as a program analyzer, a path selector, and a test data generator. Based on program execution the approaches were classified and determined the selection of test data for different level of coverage. From the research it is found that test data generation mainly concentrates on structural rather than functional (Deville, 2003; McMinn, 2004; Zhan, 2005).

Based on the survey (Rohaida, 2014) of automatic test data generation, it is found that most of the test data generation is focused on the white box testing and the coverage metric was applied for path coverage. In this study, the generated test data will cover for both coverage criteria which are path coverage and Modified Condition/Decision Coverage (MC/DC).

The latest study was done by Monpratarchai *et al.* (2014) regarding to cover path coverage metric using symbolic execution with JavaPathFinder (JPF). The following section will describe the integration of APA and ATDG to support this study.

## 2.7 Integration of APA and ATDG

As stated in the problem statement section, there have been limited studies attempted to incorporate both APA and ATDG. APA has gained more attentions from researchers in education domain in order to assess students programming (Liang, 2009). In software testing field, there have been various automated methods for test data generation. Unfortunately, APA seldom adopts these methods. Structural test data can be generated using static methods or dynamic methods. The techniques can be classified as random test data generation or path oriented test data generation, goal oriented test data generation and data specifications test data generation (Korel, 1990).

A study by Ihantola (2006) has been done in order to generate test data for structural testing in APA. In this study, Java PathFinder (JPF) software model checker was used to get test data using symbolic execution technique. Although, this proposed work integrated test data generation with APA but did not focus on covering structural code coverage in terms of test adequacy coverage criteria such as MC/DC.

Tillmann *et al*., (2013) has presented a study that helps introductory programming course as well as software engineering courses for students based on interactive-gaming teaching and learning. This study used dynamic symbolic execution technique in order to generate test data for structural testing. This study, also utilized dynamic symbolic execution that is similar as proposed by Ihantola (2006) but did not cover structural code coverage.

A study by Rohaida (2014) has proposed a test data generation framework for APA which covered both testing category: functional and structural testing. In this study, positive testing and negative testing technique has been used in order to generate test data. Although this study focuses structural testing coverage but it has proposed only path coverage testing criteria.

As shown in Table 2.10, there are only three studies on test data generation in APA that focus on structural testing.

Table 2.10 Integration of APA and ATDG (extended from Rohaida, 2014)

| Author/ Year | Testing Type | | Method | Test Data Generation Technique | Quality Factor |
|---|---|---|---|---|---|
| | Functional | Structural | | | |
| Ihantola (2006) | No | Yes | Dynamic | Symbolic execution with Java PathFinder | Correctness |
| Tilmann *et al.*, (2013) | No | Yes | Dynamic | Dynamic Symbolic execution with Pex | Correctness |
| Rohaida (2014) | Yes | Yes | Dynamic | Positive testing and Negative testing criteria | Correctness |

Based on the above discussion it is clear that to date only few studies have attempted to integrate both APA and ATDG. In addition, the past studies merely used the technique that fully covers test data adequacy criteria for student's program assessment. The above past studies merely used test adequacy criteria in structural code coverage for APA. Thus, in terms of covering structural code coverage there are limitations in integration of ATDG with APA.

By addressing these gaps, this study proposes a method called DyStruc-TDG which included path coverage and Modified Condition/Decision Coverage (MC/DC) criteria in order to generate adequate test data to cover more thorough testing coverage that significantly contributes in reducing lecturer's workload.

## 2.8 Summary

This chapter has discussed the existing work which is related to this study. It reviewed in this chapter, namely software testing, test adequacy criteria, APA, ATDG and integration of APA and ATDG. In order to support this study all the necessary justification has been provided mainly focused on automatic programming assessment and its integration with test data generation for structural testing in academic.

# CHAPTER 3

# METHODOLOGY

This chapter explains the research methodology that has been used in this study in order to achieve the research objectives and answer research questions which is formulated in **Chapter 1.** This chapter presents the detail research procedures. This includes the steps taken and the methods used to formulate research problem until evaluating the proposed work.

## 3.1 Research Procedures

This study is conducted guiding by the research procedure as shown Figure 3.1.This research procedure consists of four phase's namely theoretical study, construction of method, development of prototype and evaluation and conclusion. It also describes the activity, methods and expected outcome for each phases.

Table 3.1 Research Procedure

| No | Phase | Activity | Method | Expected Outcomes |
|---|---|---|---|---|
| 1 | Theoretical Study | -Searching for issues<br>-Investigation of earlier work<br>-Explore possible solution | Literature Survey | Proposal |
| 2 | Construction of Method | -Construct test data generation method by translating the selected structural coverage criteria into test cases design to inline with the identified types of control structure. | -Positive testing for structural coverage (path coverag +MC/DC coverage) | DyStruc-TDG [RO-1] |
| 3 | Development of Prototype | -Develop a working application as a test data generator<br><br>- Perform unit testing | -Prototyping<br>-Black-box testing | Working Prototype (test data generator) |
| 4 | Evaluation and Conclusion | -Evaluate the prototype<br>-Analysis and interpreting | -Controlled experiment [one-group pre-test and post-test design]<br>-Comparative Evaluation | Evaluation measures<br><br>Findings [RO-2] |

### 3.1.1 Theoretical Study

A theoretical study consists of concepts, together with their definitions and existing theories that are used for particular study (Sekaran, 2003). Through theoretical study, it helps to understand the problems and finally it directs to formulate research objectives and questions as well. The method used in this phase is to review the literature survey that involves automatic programming assessment, automatic test data generation and structural code coverage. The outcome of this theoretical study is proposal.

### 3.1.2 Construction of Method

In this study, the method is constructed by translating the selected structural coverage criteria into design of test cases. This method is verified in order to measure the test data adequacy criteria by doing positive testing. The output of this phase is DyStruc-TDG. In this study, two test adequacy coverage criteria have been adopted to construct the test data generation method which is as follows:

1. Path coverage
2. Modified Condition/ Decision Coverage (MC/DC)

The following Figure 3.1 shows the process of DyStruc-TDG method which generates test data for path coverage and MC/DC coverage criteria.

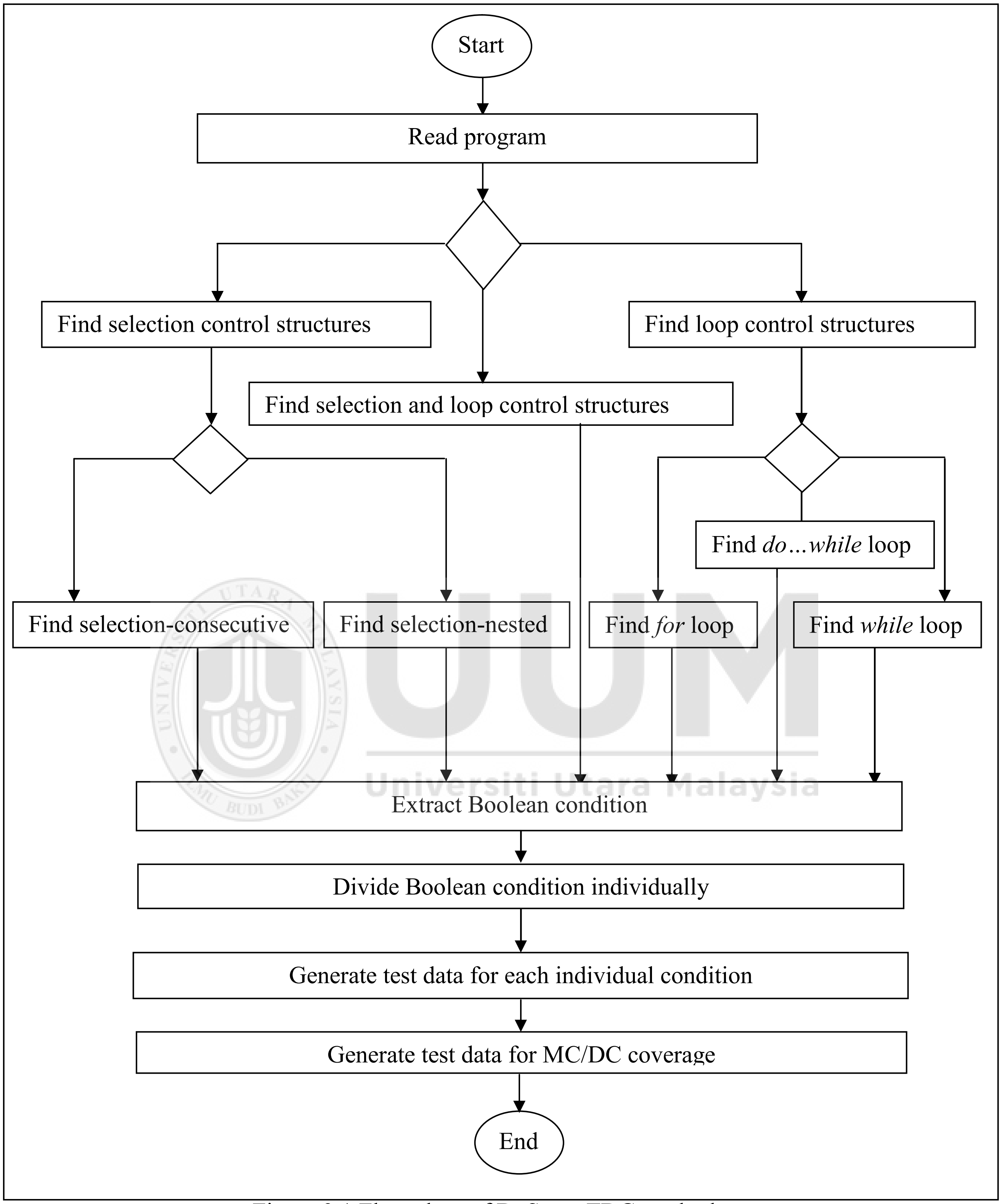

Start
Read program
Find selection control structures
Find selection and loop control structures
Find loop control structures
Find *do…while* loop
Find selection-consecutive
Find selection-nested
Find *for* loop
Find *while* loop
Extract Boolean condition
Divide Boolean condition individually
Generate test data for each individual condition
Generate test data for MC/DC coverage
End

Figure 3.1 Flow chart of DyStruc-TDG method

### 3.1.3 Development of Prototype

In this step a prototype has been developed to implement the test data generator method. The technique is used here is the prototyping technique (Sommerville, 1995; Pressman, 1997). This prototype will help the users to test the requirements. A unit testing has performed in this prototype development phase in order to identify errors of the system. The output of this phase is test data generator.

### 3.1.4 Evaluation and Conclusion

Evaluation and conclusion is the last phase of this study. It measures the test data generation method whether or not it improves the completeness coverage of reliability - test data adequacy criteria. The evaluation with regard to this aim is purposely to achieve the second research objective (**RO-2**). The research in this study has divided into two parts: a Controlled experiment and Comparative evaluation. The comparative evaluation is adopted from Rohaida (2014).

#### 3.1.4.1 Controlled Experiment

A controlled experiment that utilizes the one-group pretest and post-test design is conducted in this study. It is an experimental design in which a single group is measured or observed before and after being exposed to a treatment (Fraenkel and Wallen, 2000). This experiment is designed to the test data adequacy of DyStruc-TDG method.

#### 3.1.4.2 Procedures of Experiment

Procedures of the controlled experiment are divided into two experiments namely *pre-test* and *post-test* experiment.

**(i) The procedures of *pre-test* experiment**

In *pre-test* experiment, the subjects were explained around ten minutes regarding the experiment. They were explained about the objective, tasks to do, and definition of technical terms. After explanation they were provided a sample programming exercises and they required to view and understand it. At the same time, they were given also solutions of sample programming exercises. Finally, the test result was recorded in the provided *pre-test* question.

**(ii) The procedures of *post-test* experiment**

This procedure was similar to *pre-test* experiment, except in *post-test* the DyStruc-TDG method was used. Time and questions were same as pre-test experiment. The questions were distributed to each subject and collected by hand.

### 3.1.4.3 Subject of Experiment

The subject of the controlled experiment was the lectures who have been teaching the programming courses at least one semester at UUM. Due to different teaching schedules of the lecturers the experiment is conducted as multi-shot sessions individually. The data is collected from the subjects only at one time instead of collecting in several times. In this experiment, the number of subjects was ten (10). In this study, the extent of researcher interference was minimal. The subject system was the tool developed as a working application for DyStruc-TDG or test data generator.

### 3.1.4.4 Instrument of Data Collection

The purpose of this study was to evaluate the completeness of coverage of DyStruc-TDG method. The experiment consists of two sets of *pre-test* and *post-test* questions in order to collect data. The aim was to test the correctness of test case coverage for structural test data generation in APA. The experiment consists of *pre-test* and *post-test* questions (refer to **Appendix B**).

The set of *pre-test* and *post-test* questions are consisted of the same contents. Four samples of programming exercises were used as the assignments which of covering the main two control structures (selection and repetition) included in any of introductory programming courses. One question to cover the selection, two questions with regard repetition (counter-loop and sentinel-loop) and the remaining one is an integration of selection and repetition control structures. Each exercise was provided with solution model (refer to **Appendix A**).

### 3.1.4.5 Comparative Evaluation

Comparative evaluation also presented in this study to compare in terms of test data adequacy for structural testing. The comparison of DyStruc-TDG was done among three studies in structural testing namely Ihantola (2006) and Tillmann *et al.*, (2013); Rohaida (2014) and DyStruc-TDG. A sample of programming exercise is used for this comparison. Their detailed of comparative evaluation will be further discussed in **Chapter 5**.

### 3.1.5 Threats to the Validity of the Experiment

In order to ensure the adequate validity for the experiment it is required to design experiment carefully to minimize the threats. There are several types of threats to the validity of experiment in software engineering context (Wohlin *et al.,* 2000). The related threats are discussed as below.

#### 3.1.5.1 Internal Validity

Theoretically, threats to internal validity are concerned with issues that can affect the independents variable with respect to causality or causal relationship. They are with regard to factors that can make experiment show the behavior that is not due to the treatment but to the interference factor.

The threat regarding the internal validity is probably causing the reveal of the samples of programming exercises used in the experiment. In this experiment, the subjects could not be quarantined in one laboratory session because the constraints of having various teaching and consultation schedules. Therefore some subjects might have some hints to the questions prior to their turn. In order to minimize this problem, they were personally contacted to be in the schedule of the experiment. The schedule of the experiment was planned within a short period of time. The researcher was notified about confidentiality of the content of the experiment. In term of subjects selection for the experiment were not a problem to the internal validity because all of the targeted subjects participated in the experiment.

#### 3.1.5.2 External Validity

The external validity is concerned with generalization and their threats related to the ability to generalize experiment results outside the experiment setting. The main threat to the external validity Controlled Experiment concerns the generalization of the setting. This is with regard to the samples of programming exercises used while conducting experiment. Practically it is impossible to cover all the topics of the course syllabus in one experiment with limited time allocated for each subject. Since the major aim of the introductory programming course is to ensure students to understand and apply basic concepts of programming. In term of generalization of subjects, this threat does not affect the external validity, as it is highly probable that similar results should be obtained when running the experiment.

#### 3.1.5.3 Construct Validity

Construct validity refers to the degree of the options used which are measured properly. In controlled experiment used valid metrics to measure the criteria of test data adequacy (Rohaida, 2014). In term of the evaluations with regard to the effectiveness the questionnaires has been provided.

## 3.2 Summary

In summary, this chapter has explained in details the research procedures that carried out throughout the study. The procedures have described about the four phase's namely theoretical study, construct method, prototype development and evaluation and conclusion. Besides, this study also emphasizes more on controlled experiment which is consisted of *pre-test* and *post-test* experiment.

# CHAPTER 4

# PROPOSED METHOD (DyStruc-TDG)

This chapter provides the explanation of the proposed work for this study which is so-called DyStruc-TDG. DyStruc-TDG is a method to generate test data which includes an adequate set of test data to perform the dynamic structural testing of a program for APA. DyStruc-TDG is designed based on the result of the literature surveys and previous preliminary study.

## 4.1 Structural Test Data Generation

In the following section selection control structures and loop control structures will be described in details. The discussion will also cover the path and Modified Condition/Decision Coverage (MC/DC) coverage for both control structures.

### 4.1.1 Selection Control Structure (Path Coverage)

According to Liang (2009) Java programming consists of *if*, *if…else* and *switch…case* decision making statements in selection control structures. Selection control structures involve consecutive (sequential) or nested structure. In this study, the both control structures are covered. The following section will describe the derived test cases for selection (consecutive and nested) control structures.

#### (i) Selection - consecutive (sequential)

In order to derive test cases for selection control structures, it depends on some properties (refer to **Chapter 3, Section** 3.1.2 (i)). In this study option refers to condition in selection and loop control structures.

Considering the tested program has two selection control structures (Selection 1 and Selection 2) and selection 1 consists of 2 options and selection 2 consists of 3 options.

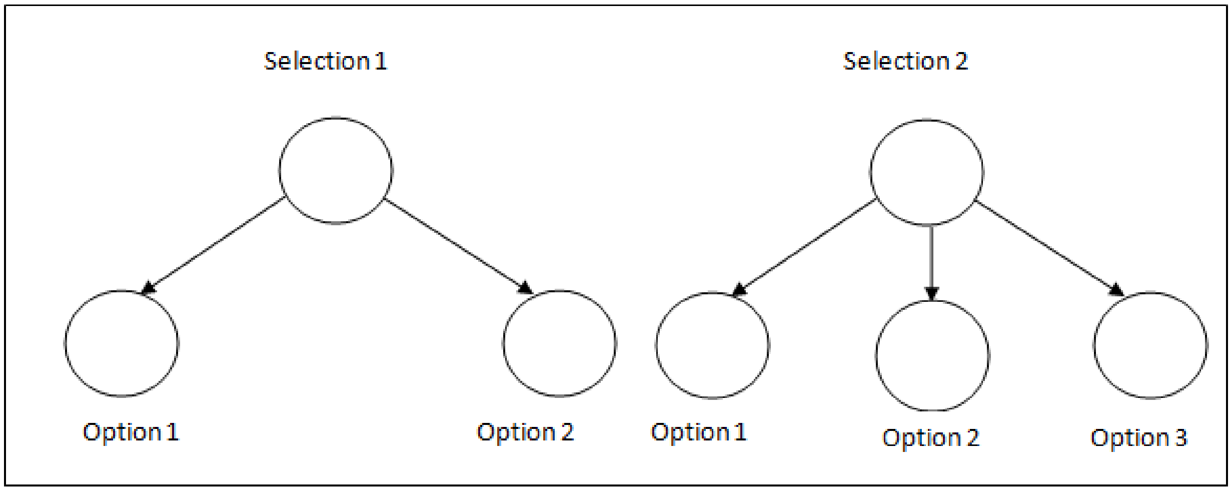


Figure 4.1 Example of tree structure for selection control structures

If a lecturer considers valid and invalid input conditions, then the Table 4.1 shows the number of derived test cases.

Table 4.1 Generated test cases based on valid and invalid input conditions

| **Test Case** | **Input Conditions** |
|---|---|
| TC 1 | Selection 1- Option 1-Valid |
| TC 2 | Selection 1- Option 2-Valid |
| TC 3 | Selection 1-Invalid |
| TC 4 | Selection 2- Option 1-Valid |
| TC 5 | Selection 2- Option 2-Valid |
| TC 6 | Selection 2- Option 3-Valid |
| TC 7 | Selection 2-Invalid |

If a lecturer considers only valid input conditions, then the Table 4.2 shows the number of derived test cases.

Table 4.2 Generated test cases based on valid input conditions

| Test Case | Input Conditions |
|---|---|
| TC 1 | Selection 1- Option 1-Valid |
| TC 2 | Selection 1- Option 2-Valid |
| TC 3 | Selection 2- Option 1-Valid |
| TC 4 | Selection 2- Option 2-Valid |
| TC 5 | Selection 2- Option 3-Valid |

**(ii) Selection – Nested**

In nested selection control structure, there is more than one selection control structure which consists of parent selection control structures and inside child selection control structures. In parent control structures and child control structures might have more than one option.

For example, if there is three selection control structures (Selection 1, Selection 2, and Selection 3) exist the tree can be visualized as follows:

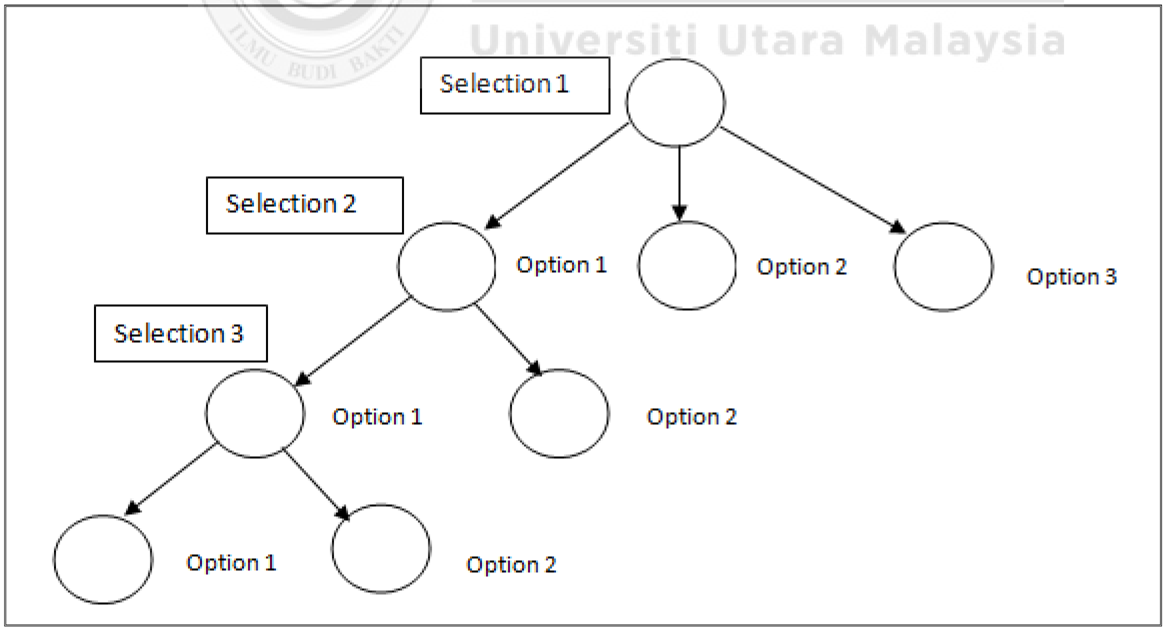


Figure 4.2 Example of tree structure for nested selection control structures

Based on the above tree structures, Table 4.3 shows the following number of derived test cases for all valid input conditions.

Table 4.3 Test cases for valid input conditions

| Test Case | Input Condition |
|---|---|
| TC 1 | Selection 1-Option 1-Valid, Selection 2-Option 1-Valid, Selection 3-Option 1-Valid |
| TC 2 | Selection 1-Option 1-Valid, Selection 2-Option 1-Valid, Selection 3-Option 2-Valid |
| TC 3 | Selection 1-Option 1-Valid, Selection 2-Option 2-Valid |
| TC 4 | Selection 1-Option 2-Valid |
| TC 5 | Selection 1-Option 3-Valid |

Based on the tree structures, Table 4.3 shows the generated five test cases for all valid input conditions. However, in this study it includes one test case for invalid input condition. The purpose of including one invalid input condition is to reduce the total number of generated test cases. Based on the Figure 4.2, the generated test cases for invalid condition are shown in Table 4.4.

Table 4.4 Test cases for invalid input conditions

| Test Case | Input Condition |
|---|---|
| TC 1 | Selection 1-invalid |
|  | Selection 2-invalid |
|  | Selection 3-invalid |

Considering the tested program has two selection-control structures (Selection1 and Selection 2) and each selection-control structure has three options (***Option 1***, ***Option 2*** and ***Option 3***), the following Figure 4.3 describes the representation.

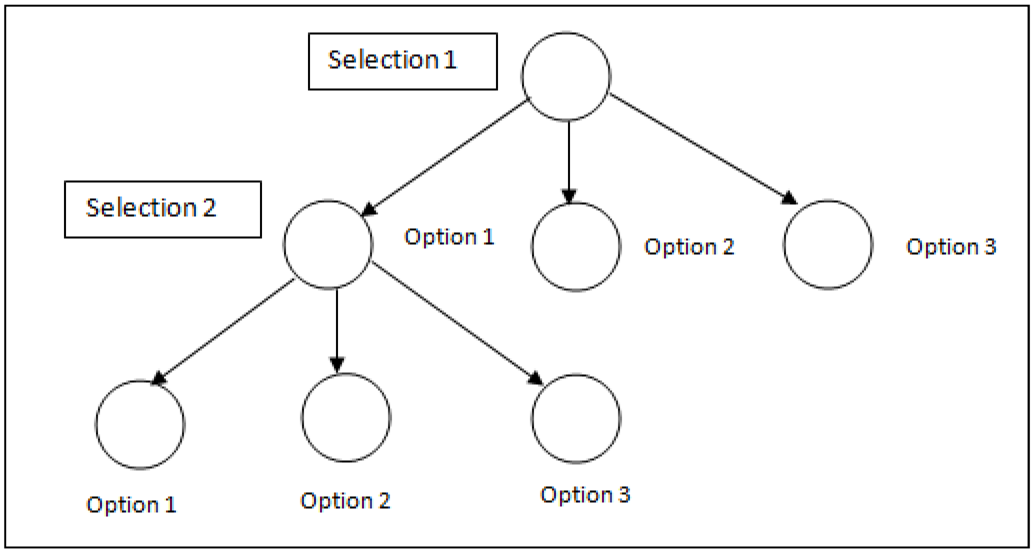


Figure 4.3 Example of two selection control structures

If a lecturer considers valid and invalid input conditions, then the Table 4.5 shows the number of derived test cases.

Table 4.5 Test cases for all input conditions for selection-nested

| Test Case | Input Conditions |
|---|---|
| TC 1 | Selection1-Option1-Valid, Selection2-Option1-Valid |
| TC 2 | Selection1-Option1-Valid, Selection2-Option2-Valid |
| TC 3 | Selection1-Option1-Valid, Selection2-Option3-Valid |
| TC 4 | Selection1-Option2-Valid |
| TC 5 | Selection1-Option3-Valid |
| TC 6 | Selection1-Invalid, Selection2-Invalid |

Based on Table 4.5 there are six generated test cases where five test cases are valid (TC1, TC2, TC3, TC4, TC5) and one of them is invalid (TC6).

If a lecturer considers only valid input conditions, then the Table 4.6 shows the number of derived test cases.

Table 4.6 Test cases for only valid input conditions for selection-nested

| Test Case | Input Conditions |
|---|---|
| TC 1 | Selection1-Option1-Valid |
| TC 2 | Selection1-Option2-Valid |
| TC 3 | Selection1-Option3-Valid |
| TC 4 | Selection2-Option1-Valid |
| TC 5 | Selection2-Option2-Valid |
| TC 6 | Selection2-Option3-Valid |

Table 4.6 shows that there are six test cases are generated where invalid input conditions are excluded.

### 4.1.2 Loop control Structure (Path Coverage)

According to Liang (2009) Java programming consists of three loops namely *for* loops, *while loops* and *do-while* loops. The while loops are similar with do while loops. It is different only in terms of execution. Besides, these two loops there is another loop called *for* loop. *For* loop is used as a counter.

Considering three loops (*Loop1*, *Loop 2* and *Loop 3*) for the program and including valid and invalid input conditions, the following Table 4.7 shows the generated test cases.

Table 4.7 Test cases for valid and invalid input conditions for loop control structures

| Test Case | Input Conditions |
|---|---|
| TC 1 | Loop1-Valid |
| TC 2 | Loop1-Invalid |
| TC 3 | Loop2-Valid |
| TC 4 | Loop2-Invalid |
| TC 5 | Loop3-Valid |
| TC 6 | Loop3-Invalid |

In terms of including only valid input conditions, the following Table 4.8 shows the generated test cases.

Table 4.8 Test cases for valid input for loop control structures

| **Test Case** | **Input Conditions** |
|---|---|
| TC 1 | Loop1-Valid |
| TC 2 | Loop2- Valid |
| TC 3 | Loop3-Valid |

The next section will describe in details of the Modified Condition/Decision Coverage (MC/DC) test adequacy coverage criteria for both control structures (Selection and Loop).

### 4.1.3 Selection Control Structure (Modified Condition/Decision Coverage)

In order to implement Modified Condition/Decision Coverage (MC/DC) test adequacy coverage criteria first need to consider the possible outcome which is true and false for each option. In order to derive test cases for MC/DC coverage it relies on some properties. The properties are discussed in **chapter 2, section 2.3.4**. The explanation of the test cases design for MC/DC coverage will be discussed in the following section.

**(i)** If a lecturer considers ***one (1) option,*** in one selection control structures; the formula of the truth table is $N^{2,}$ where N is the number of option (N=1, 2, 3, 4, …, n). In this case, the number of condition is one:

Table 4.9 Truth table for one option

| Test Case | Option (A) |
|---|---|
| TC 1 | True |
| TC 2 | False |

Based on MC/DC test adequacy criteria concept, the following Table 4.10 shows the generated test cases considering independently effect of one option.

Table 4.10 Generated test cases for one option

| Option (A) | Pattern |
|---|---|
| True | 1 |
| False | 2 |

**(ii)** If a lecturer considers ***two (2) options,*** in one selection control structure (Selection 1) and this selection control structure consists of 2 options; the truth table for two options is as follows:

Table 4.11 Truth table for two options

| Test Case | Option (A) | Option (B) | Option (A&&B) |
|---|---|---|---|
| TC 1 | True | True | True |
| TC 2 | True | False | False |
| TC 3 | False | True | False |
| TC 4 | False | False | False |

For the above two options truth table the generated test cases for MC/DC coverage is described below. The following Tables will describe the generated test cases for independently effect of each option.

In this case considering independently effect on ***Option (A)***, Table 4.12 shows the produced result.

Table 4.12 Independently effect on option (A)

| Option (A) | Option(B) | Option (A&&B) | Pattern |
|---|---|---|---|
| True | True | True | 1 |
| False | True | False | 2 |

In this case considering independently effect on ***Option (B)***, Table 4.13 shows the produced result.

Table 4.13 Independently effect option (B)

| Option (A) | Option (B) | Option (A&&B) | Pattern |
|---|---|---|---|
| True | True | True | 1 |
| True | False | False | 3 |

The following Table 4.14 produces the final result by combining the test cases from the above both tables. In this case ***Option (A)*** and ***Option (B)*** are considered independently. Thus, applying MC/DC coverage criteria concept it reduces the number of test cases from 4 (see **Table 4.11**) to 3.

Table 4.14 Independently effect option (A) and (B)

| Option (A) | Option(B) | Option (A&&B) | Pattern |
|---|---|---|---|
| True | True | True | 1 |
| False | True | False | 2 |
| True | False | False | 3 |

**(iii)** If a lecturer considers **three (3)** ***options,*** in one selection control structure (Selection 1) and it consists of 3 options; the Table 4.15 shows the truth table for three options is as follows:

Table 4.15 Truth table for three options

| Test Case | Option (A) | Option (B) | Option (C) | Option (A&&B&&C) |
|---|---|---|---|---|
| TC 1 | True | True | True | True |
| TC 2 | True | True | False | False |
| TC 3 | True | False | True | False |
| TC 4 | True | False | False | False |
| TC 5 | False | True | True | False |
| TC 6 | False | True | False | False |
| TC 7 | False | False | True | False |
| TC 8 | False | False | False | False |

For the above three options; ***Option (A), Option (B)*** and ***Option (C)*** from the truth table, the generated test cases for MC/DC coverage is described below. The following Tables will describe the generated test cases for independently effect of each option.

In this case considering independently effect on ***Option (A)***, Table 4.16 shows the produced result.

Table 4.16 Independently effect option (A)

| Test Case | Option (A) | Option (B) | Option (C) | Option(A&&B&&C) |
|---|---|---|---|---|
| TC 1 | True | True | True | True |
| TC 5 | False | True | True | False |

In this case considering independently effect on ***Option (B)***, Table 4.17 shows the produced result.

Table 4.17 Independently effect option (B)

| Test Case | Option (A) | Option (B) | Option (C) | Option (A&&B&&C) |
|---|---|---|---|---|
| TC 1 | True | True | True | True |
| TC 3 | True | False | True | False |

In this case considering independently effect on ***Option (C)***, Table 4.18 shows the produced result.

Table 4.18Independently effect option (C)

| Test Case | Option (A) | Option (B) | Option (C) | Option (A&&B&&C) |
|---|---|---|---|---|
| TC 1 | True | True | True | True |
| TC 2 | True | True | False | False |

The following Table 4.19 produces the final result by combining the test cases from the above Tables. In this case ***Option (A)***, ***Option (B)*** and ***Option (C)*** are considered independently. Thus, applying MC/DC coverage criteria concept it reduces the number of test cases from 8 (**Table 4.15**) to 4.

Table 4.19Independently effect option (A), (B) and (C)

| Test Case | Option (A) | Option (B) | Option (C) | Option (A&&B&&C) |
|---|---|---|---|---|
| TC 1 | True | True | True | True |
| TC 5 | False | True | True | False |
| TC 3 | True | False | True | False |
| TC 2 | True | True | False | False |

**(iv)** If a lecturer considers **four (4)** ***options,*** in one selection control structure (Selection 1) and if this selection control structure consists of 4 options ***Option (A), Option (B), Option (C)*** and ***Option (D)***; the truth table for four options is as follows:

Table 4.20 Truth table for four options

| Test Case | Option (A) | Option (B) | Option (C) | Option (D) | Option (A&&B&&C&&D) |
|---|---|---|---|---|---|
| TC 1 | True | True | True | True | True |
| TC 2 | True | True | True | False | False |
| TC 3 | True | True | False | True | False |
| TC 4 | True | True | False | False | False |
| TC 5 | True | False | True | True | False |
| TC 6 | True | False | True | False | False |
| TC 7 | True | False | False | True | False |
| TC 8 | True | False | False | False | False |
| TC 9 | False | True | True | True | False |
| TC 10 | False | True | True | False | False |
| TC 11 | False | True | False | True | False |
| TC 12 | False | True | False | False | False |
| TC 13 | False | False | True | True | False |

| Test Case | Option (A) | Option (B) | Option (C) | Option (D) | Option (A&&B&&C&&D) |
|---|---|---|---|---|---|
| TC 14 | False | False | True | False | False |
| TC 15 | False | False | False | True | False |
| TC 16 | False | False | False | False | False |

For the above four options; ***Option (A), Option (B), Option (C)*** and ***Option (D)*** from the truth table, the generated test cases for MC/DC coverage is described below. The following Tables will describe the generated test cases for independently effect of each option.
In this case considering independently effect on ***Option (A)***, Table 4.21 shows the produced result.

Table 4.21 Independently effect ption (A)

| Test Case | Option (A) | Option (B) | Option (C) | Option (D) | Option (A&&B&&C&&D) |
|---|---|---|---|---|---|
| TC 1 | True | True | True | True | True |
| TC 9 | False | True | True | True | False |

In this case considering independently effect on ***Option (B)***, Table 4.22 shows the produced result.

Table 4.22 Independently effect option (B)

| Test Case | Option (A) | Option (B) | Option (C) | Option (D) | Option (A&&B&&C&&D) |
|---|---|---|---|---|---|
| TC 1 | True | True | True | True | True |
| TC 5 | True | False | True | True | False |

In this case considering independently effect on ***Option (C)***, Table 4.23 shows the produced result.

Table 4.23Independently effect option (C)

| Test Case | Option (A) | Option (B) | Option (C) | Option (D) | Option (A&&B&&C&&D) |
|---|---|---|---|---|---|
| TC 1 | True | True | True | True | True |
| TC 3 | True | True | False | True | False |

In this case considering independently effect on ***Option (D)***, Table 4.24 shows the produced result.

Table 4.24Independently effect option (D)

| Test Case | Option (A) | Option (B) | Option (C) | Option (D) | Option (A&&B&&C&&D) |
|---|---|---|---|---|---|
| TC 1 | True | True | True | True | True |
| TC 2 | True | True | True | False | False |

The following Table 4.25 produces the final result by combining the test cases from the above Tables. In this case ***Option (A)***, ***Option (B), Option (C)*** and ***Option (D)*** are considered independently. Thus, applying MC/DC coverage criteria concept it reduces the number of test cases from 16 (see **Table 4.20**) to 5.

Table 4.25 Independently effect option (A), (B), (C) and (D)

| Test Case | Option (A) | Option (B) | Option (C) | Option (D) | Option (A&&B&&C&&D) |
|---|---|---|---|---|---|
| TC 1 | True | True | True | True | True |
| TC 9 | False | True | True | True | False |
| TC 5 | True | False | True | True | False |
| TC 3 | True | True | False | True | False |
| TC 2 | True | True | True | False | False |

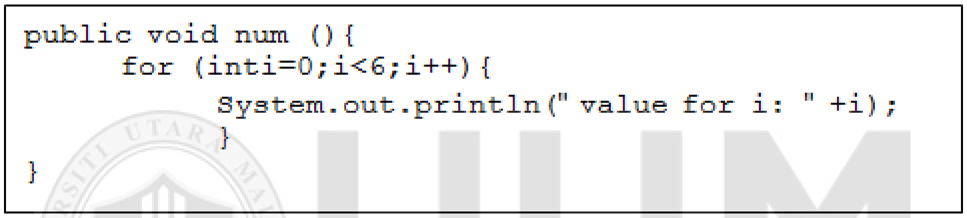
The above **Section 4.1.3** has explained in details about MC/DC coverage criterion for selection control structures. In this case, four options have discussed in order to show the consistency of generating test cases for MC/DC coverage criterion. The explanation shows from the truth table that the number of test cases by applying MC/DC coverage criteria has reduced the number of test cases. For one option it has only two values (true and false) so the number of test cases for one option is 2. If there are two options the truth table shows 4 combinations of test cases; but applying MC/DC coverage it reduces to 3 possible test cases, same applies to number of option three and four. It generates number of test cases 4 and 5 respectively.

Finally, it concludes from the explanation above that the formula for generating test cases by applying MC/DC test adequacy coverage criteria is (N+1) where N is the number of option. For example,  if there is 2 conditions by applying MC/DC the number of generated test cases will be (2+1)=3. The following Table 4.26 shows the formula for number of conditions involve in a program.

### 4.1.4 Loop Control Structure (Modified Condition/Decision Coverage)

In this section will be discussed about three types of loop control structures as mentioned in **Chapter 3**, **Section** 2.3.4(ii) for MC/DC coverage criterion.

If a lecturer considers ***for*** **loop,** in a program has one ***for loop*** control structure (loop 1) such as below:



```
public void num (){
      for (inti=0;i<6;i++){
            System.out.println(" value for i: " +i);
            }
}
```

Figure 4.4 Example of *for loop* control structure

In this case, the number of option is one. The formula of truth table is $N^{2,}$ where N is the number of option (N=1, 2, 3, 4….n). The truth table for one option is as follow:

Table 4.26 Truth table for one option (loop)

| Test Case | Option (A) |
|---|---|
| TC 1 | True |
| TC 2 | False |

The following Table 4.27 shows the generated test cases considering independently effect of one option. In this case, one option has only two values in a decision either true or false.

Table 4.27 Generated test cases for one option (loop)

| Option (A) | Pattern |
|---|---|
| True | 1 |
| False | 2 |

**(ii)** If a lecturer considers ***do…while*** loop, in this program has one ***do…while*** loop control structure (loop 1); and it consist of 2 options (***Option X*** and ***Option Y***).

```
do{
     //Some statements
}
while (x>20&&y<50);
```

Figure 4.5 Example of *do…while* loop control structure

In this case, the number of option is two. The formula of truth Table is $N^{2,}$ where N is the number of option (N=1, 2, 3, 4... n). The truth table for one option is as follows:

Table 4.28 Truth table for two options (loop)

| Test Case | Option (X) | Option (Y) | Option(X&&Y) |
|---|---|---|---|
| TC 1 | True | True | True |
| TC 2 | True | False | False |
| TC 3 | False | True | False |
| TC 4 | False | False | False |

The following Tables show the generated test cases considering independently effect of each option.

In this case considering independently effect on ***Option (X)***, Table 4.30 shows the produced result.

Table 4.29 Independently effect option (X)

| Option (X) | Option (Y) | Option(X&&Y) | Pattern |
|---|---|---|---|
| True | True | True | 1 |
| False | True | False | 2 |

In this case considering independently effect on ***Option (Y)***, Table 4.31 shows the produced result.

Table 4.30 Independently effect option (Y)

| Option (X) | Option (Y) | Option(X&&Y) | Pattern |
|---|---|---|---|
| True | True | True | 1 |
| True | False | False | 3 |

The following Table 4.31 produces the final result by combining the test cases from the above Tables. In this case ***Option (X)*** and ***Option (Y))*** are considered as independently. Thus, applying MC/DC coverage criteria concept it reduces the number of test cases from 8 (**Table 4.29**) to 3.

Table 4.31Independently effect option (X) and option (Y)

| Option (X) | Option (Y) | Option(X&&Y) | Pattern |
|---|---|---|---|
| True | True | True | 1 |
| False | True | False | 2 |
| True | False | False | 3 |

**(iii)** If a lecturer considers ***while loop***, in term condition for *while loop* is same as *do…while* loop. This kind of loop structure is described in *do…while* loop.

The generated number of test cases for nested loop for MC/DC is also same as described above.

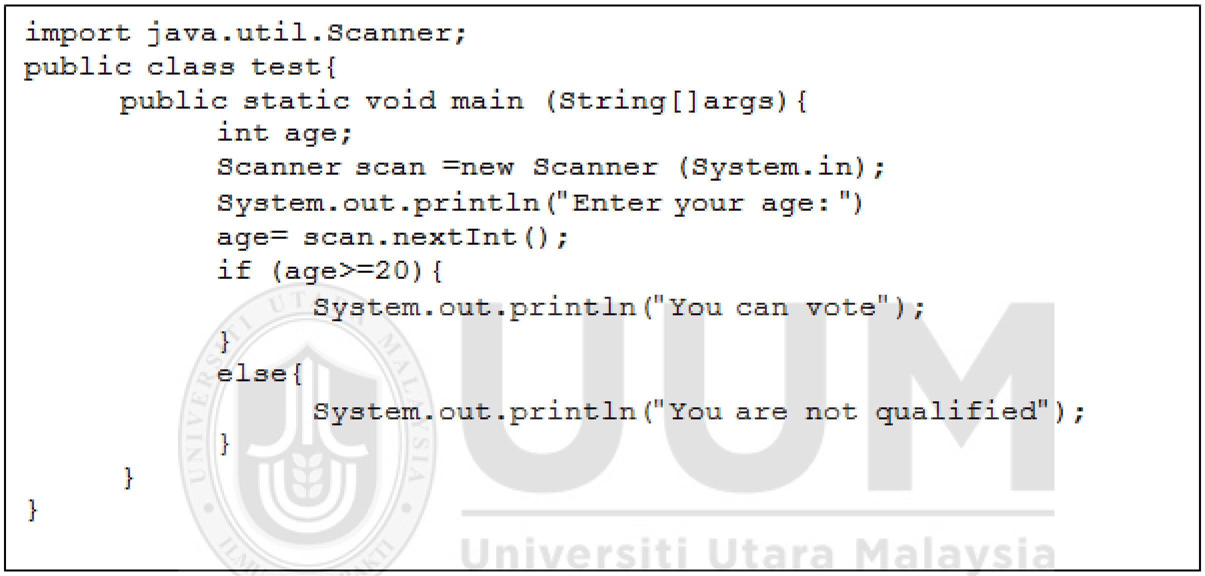
The next section will explain the generated test cases for MC/DC coverage criteria by using programming example.

## 4.2 Structural Test Data Generation by Examples

In this section some example will be used in order to relate structural test data generation with APA.

### 4.2.1 Selection Control Structure

**(i) One (1) option**: An example of programming exercise with one option (age>=20)



```
import java.util.Scanner;
public class test{
      public static void main (String[]args){
            int age;
            Scanner scan =new Scanner (System.in);
            System.out.println("Enter your age:")
            age= scan.nextInt();
            if (age>=20){
                  System.out.println("You can vote");
            }
            else{
                  System.out.println("You are not qualified");
            }
      }
}
```

Figure 4.6 Sample of programming exercise with one option

Table 4.25 illustrates the sample of programming exercise with its expression, option and true false value.

Table 4.32 Illustration of sample program

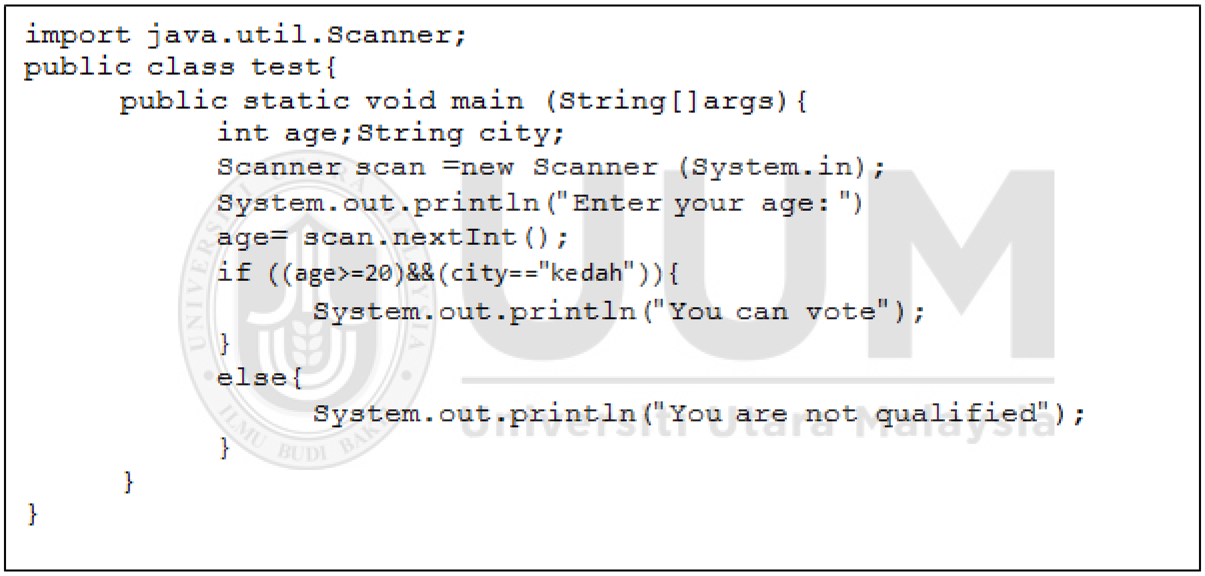
| Expression | Option | True Value | False Value |
|---|---|---|---|
| age>=20 | age | 20 | 18 |

By applying MC/DC test adequacy coverage criteria it generates 2 (two) test cases which are shown below:

Table 4.33 Generated test cases for one option (MC/DC)

| Test Case | Value |
|---|---|
| TC 1 | **True**<br>20 |
| TC 2 | **False**<br>18 |

**(ii) Two (2) options**: An example of programming exercise with two options (age>=20&&city=="kedah")



```
import java.util.Scanner;
public class test{
      public static void main (String[]args){
            int age;String city;
            Scanner scan =new Scanner (System.in);
            System.out.println("Enter your age: ")
            age= scan.nextInt();
            if ((age>=20)&&(city=="kedah")){
                  System.out.println("You can vote");
            }
            else{
                  System.out.println("You are not qualified");
            }
      }
}
```

Figure 4.7 Sample of programming exercise with two options

Table 4.34 illustrates the sample of programming exercise with its expression, option and true false value.

Table 4.34 Illustration of sample program for two options

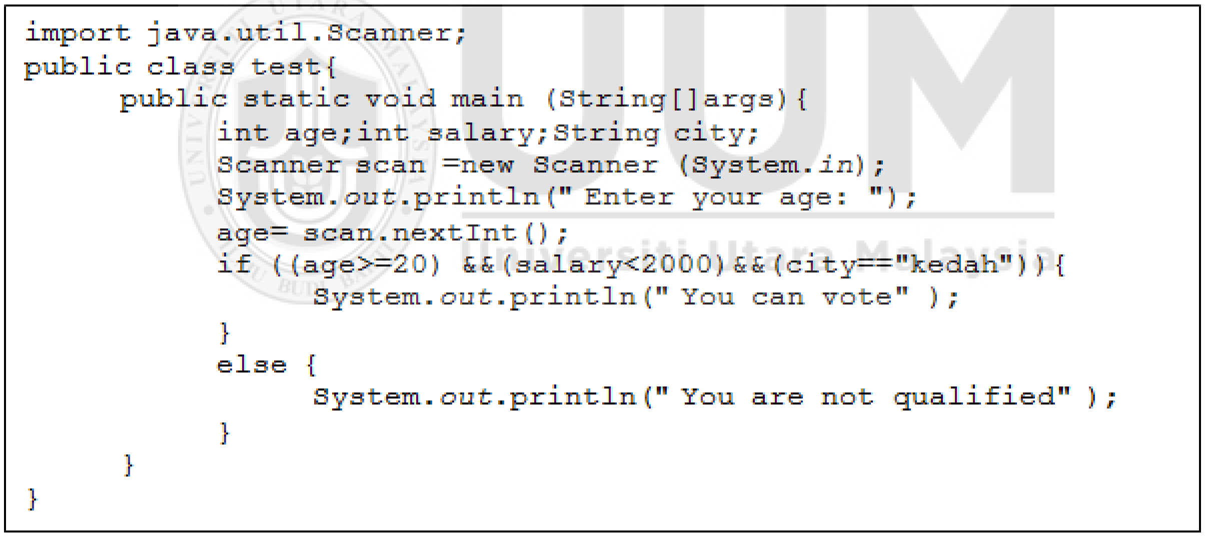
| Expression | Option | True value | False value |
|---|---|---|---|
| ((age>=20) &&(city==”kedah”)) | age | 20 | 18 |
| | city | "Kedah" | "Anything" |

**(iii) MC/DC coverage:** By applying MC/DC test adequacy coverage criterion it generates three (3) test cases which are shown below:

Table 4.35 Generated test cases for two options (MC/DC)

| Test Case | Value | |
|---|---|---|
| TC 1 | **True**<br>20 | **True**<br>"Kedah" |
| TC 2 | **False**<br>18 | **True**<br>"Kedah" |
| TC 3 | **True**<br>20 | **False**<br>"Anything" |

An example of programming exercise with three options ((age>=20&&salary< 2000)&&(city=="kedah"))



```
import java.util.Scanner;
public class test{
      public static void main (String[]args){
            int age;int salary;String city;
            Scanner scan =new Scanner (System.in);
            System.out.println(" Enter your age: ");
            age= scan.nextInt();
            if ((age>=20) &&(salary<2000)&&(city=="kedah")){
                  System.out.println(" You can vote" );
            }
            else {
                  System.out.println(" You are not qualified" );
            }
      }
}
```

Figure 4.8 Sample of programming exercise with three options

Table 4.36 illustrates the sample of programming exercise with its expression, option and true false value

Table 4.36 Illustration of sample program for three options

| Expression | Option | True value | False value |
|---|---|---|---|
| ((age>=20) &&(salary<2000)&&(city ==”kedah”)) | age | 20 | 18 |
| | salary | 1800 | 2200 |
| | city | “Kedah” | “Anything” |

By applying MC/DC test adequacy coverage criteria it generates three (3) test cases which are shown below

Table 4.37Generated test cases for three options (MC/DC)

| Test Case (TC) | Value | | |
|---|---|---|---|
| TC 1 | **True**<br>20 | **True**<br>1800 | **True**<br>“Kedah” |
| TC 2 | **False**<br>18 | **True**<br>1800 | **True**<br>“Kedah” |
| TC 3 | **True**<br>20 | **False**<br>2200 | True<br>“Kedah” |
| TC 4 | **True**<br>20 | **True**<br>1800 | **False**<br>“Anything” |

## 4.2.2 Loop Control Structure (Modified Condition/Decision Coverage)

**(i)** ***do…while*** **loop :** An example of *do…while* loop programming exercise.

```
public class test{
      public static void main (String[]args){
            int x=10;
            do{
                  System.out.println("The value of X:"+x);
                  x++;
            }
            while(x<50);
      }
}
```

Figure 4.9 Sample of programming exercise *do…while* loop with one option

Table 4.38 illustrates the sample of programming exercise with its expression, option and true false value

Table 4.38 Illustration of sample program (loop)

| Expression | Option | True Value | False Value |
|---|---|---|---|
| x<50 | x | 48 | 52 |

By applying MC/DC test adequacy coverage criteria it generates two (2) test cases which are shown below:

Table 4.39 Generated test cases for one option (loop)

| Test Case | Value |
|---|---|
| TC 1 | **True** 48 |
| TC 2 | **False** 52 |

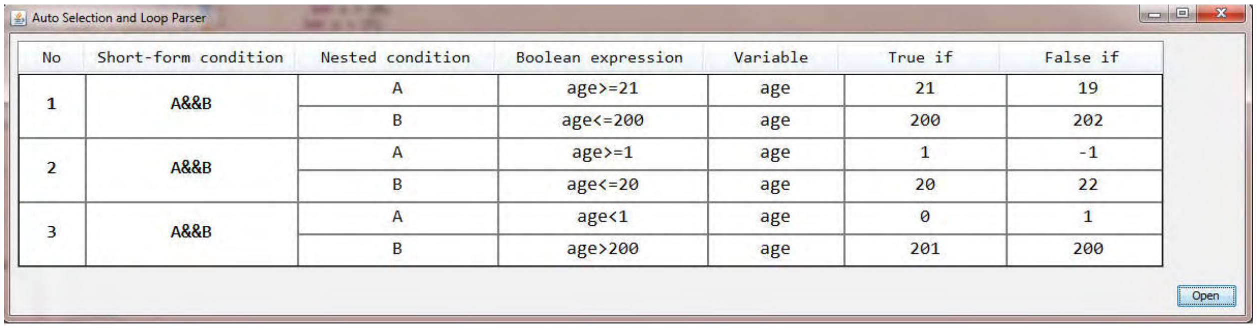
**(ii)** Generating test cases for ***while loop*** also same as described in *do… while* loop.

**(iii)** Generating test cases for ***for loop*** also same as described in *do… while* loop

In this study three types of loop control structure (*do…while* loop*, while* loop and *for* loop) has been covered to generate test data.

## 4.3 Implementation of DyStruc-TDG

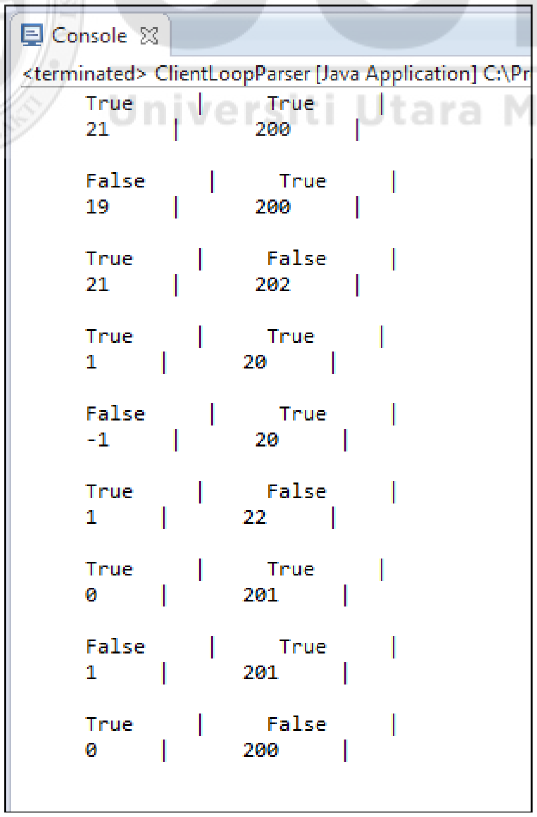
In this section, the implementation of the prototype will be described. In order to generate test data a prototype is developed to test the DyStruc-TDG method for structural testing coverage. The system specification for this prototype is used Ecilipse IDE version Juno. The method is developed by using Java programming language. The following Figures will describe in details of the implementation. The following Figure 4.10 shows an interface of running all selection control structures. The program automatically extracts the conditions written in the program and generates the test data (true and false) for each condition. Lecturer needs to click on "Open" button in order to run the students program. After clicking the "Open" button the pop up window will allow to select students program.



| No | Short-form condition | Nested condition | Boolean expression | Variable | True if | False if |
|---|---|---|---|---|---|---|
| 1 | A&&B | A | age>=21 | age | 21 | 19 |
|  |  | B | age<=200 | age | 200 | 202 |
| 2 | A&&B | A | age>=1 | age | 1 | -1 |
|  |  | B | age<=20 | age | 20 | 22 |
| 3 | A&&B | A | age<1 | age | 0 | 1 |
|  |  | B | age>200 | age | 201 | 200 |

Figure 4.10 Interface of generated test data (selection control structures)

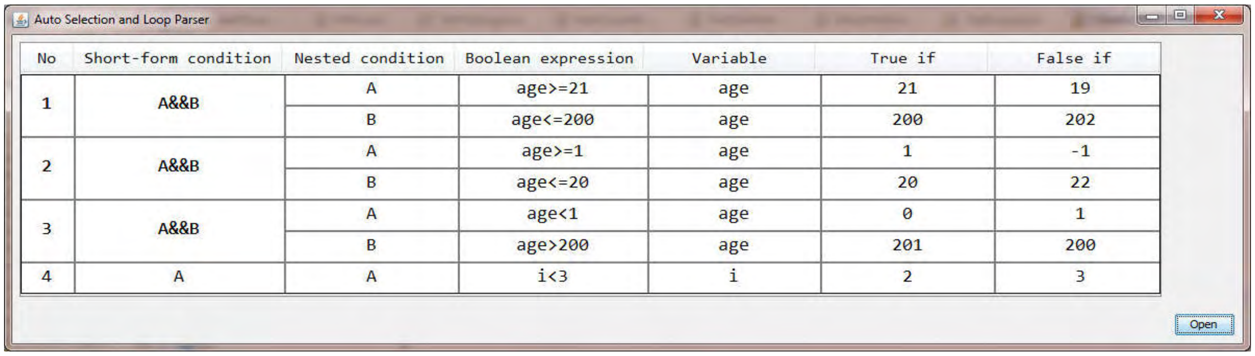
The Figure 4.11 shows the generated test cases by using the concept of coverage criteria Modified Condition/ Decision Coverage (MC/DC). In this program there are three selection control structures with two options. Based on description in section 4.4, the number of generated test cases for MC/DC is 9, where each selection control structure with two options generated 3 test cases.



```
<terminated> ClientLoopParser [Java Application] C:\Pr
     True      |     True      |
     21      |      200      |

     False      |      True      |
     19      |      200      |

     True      |      False      |
     21      |      202      |

     True      |      True      |
     1      |      20      |

     False      |      True      |
     -1      |      20      |

     True      |      False      |
     1      |      22      |

     True      |      True      |
     0      |      201      |

     False      |      True      |
     1      |      201      |

     True      |      False      |
     0      |      200      |
```

Figure 4.11 Interface of generated test data (selection control structures)

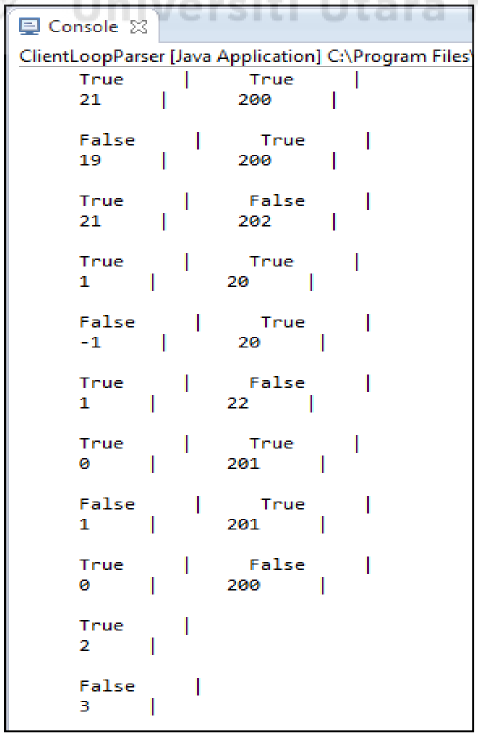
The following Figure 4.12 is an interface of running all selection control structures and loop control structures. The program automatically extracts the conditions for selection and loop control structures. The number 1 to 3 is for selection control structures and number 4 is the counter loop structures.



| No | Short-form condition | Nested condition | Boolean expression | Variable | True if | False if |
|---|---|---|---|---|---|---|
| 1 | A&&B | A | age>=21 | age | 21 | 19 |
| | | B | age<=200 | age | 200 | 202 |
| 2 | A&&B | A | age>=1 | age | 1 | -1 |
| | | B | age<=20 | age | 20 | 22 |
| 3 | A&&B | A | age<1 | age | 0 | 1 |
| | | B | age>200 | age | 201 | 200 |
| 4 | A | A | i<3 | i | 2 | 3 |

Figure 4.12 Interface of generated test data (Selection and Loop control structures)

The Figure 4.13 shows the generated test cases by using the concept of coverage criteria Modified Condition/ Decision Coverage (MC/DC). In this program there are three selection control structures with two options and one loop control structures. Based on description in section 4.4, the number of generated test cases for selection control structure is 9 and loop control structure is 2. Therefore, all together the number of generated test cases for MC/DC is 11.



```
ClientLoopParser [Java Application] C:\Program Files\
     True      |     True      |
     21      |     200      |

     False      |     True      |
     19      |     200      |

     True      |     False      |
     21      |     202      |

     True      |     True      |
     1      |     20      |

     False      |     True      |
     -1      |     20      |

     True      |     False      |
     1      |     22      |

     True      |     True      |
     0      |     201      |

     False      |     True      |
     1      |     201      |

     True      |     False      |
     0      |     200      |

     True      |
     2      |

     False      |
     3      |
```

Figure 4.13 Interface of generated test data for MC/DC (selection and loop control structures)

The next section will describe regarding the unit testing for this implementation of DyStruc-TDG.

## 4.4 Unit Testing

According to Tim Koomen and Martin Pol (1999) unit test is defined as a test which is executed by the developer in a lab in order to meet the requirements of design specification. Unit testing has been performed in this study in order to generate structural test data. Several programming exercises were used to perform unit testing to test whether or not the prototype can generate accurate test data. During unit testing the both control structures (selection and repetition) were tested.

## 4.5 Summary

This chapter provides the explanation of proposed work for the research in this study. The proposed method is called DyStruc-TDG which includes an adequate set of test data to perform dynamic structural testing of a program for APA. DyStruc-TDG covers both control structures which are selection control structures and loop control structures. The selected two test adequacy criteria which are path coverage and Modified condition/Decision coverage (MC/DC) also explained for both control structures. Besides, the implementation of DyStruc-TDG and unit testing also presented in this chapter.

# CHAPTER 5
# EVALUATION

This chapter explains the details of procedures and results found from the conducted evaluation to evaluate the DyStruc-TDG method. In this study, the evaluation consists of two parts namely Controlled Experiment and Qualitative Comparison.

The first part involves a controlled experiment to measure the positive testing of test data adequacy criteria of DyStruc-TDG. The findings of this controlled experiment will attempt to answer the second sub-research question (***RQ-2***). The details of controlled experiment have been discussed in **Chapter 3** (**Section** 3.1.4.1). In this chapter the analysis and findings from the conducted *pre-test* and *post-test* experiment will be reported.

The second part of this study explains the qualitative evaluation of DyStruc-TDG. It compares the existing tool(s) in the same domain that worked on the integration of test data generation and automatic programming assessment that focuses on the structural testing. The analysis of findings of this evaluation will be reported as well.

## 5.1 Descriptive Statistics

The following section will describe the analysis and findings from the conducted *pre-test* and *post-test* experiment. Based on conducted *pre-test* and *post-test* experiment, the following Tables present the number of test cases derived by current method and DyStruc-TDG method to test each programming exercises.

### 5.1.1 Question (1): Current method

The following Figure 5.1 shows the control flow graph of the selection control structure for Question (1).

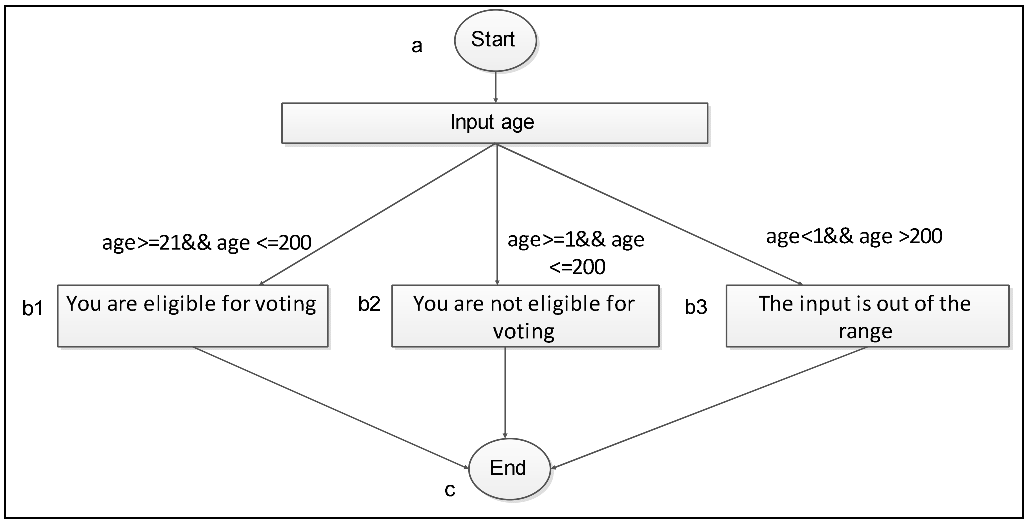

a
Start
Input age
age>=21&& age <=200
age>=1&& age <=200
age<1&& age >200
b1
You are eligible for voting
b2
You are not eligible for voting
b3
The input is out of the range
End
c

Figure 5.1 Control flow graph for selection control structure

Table 5.1 shows the paths for selection control structures which consisted of 3 paths (path 1, path 2 and path 3)

Table 5.1Number of paths for selection control structures

| Path 1 | Path 2 | Path 3 |
|---|---|---|
| (age>=21&& age <=200) | (age>=1&& age <=20) | (age<1&& age >200) |

Based on conducted *pre-test* experiment, the following Table 5.2 shows the number of derived test cases for current method. The question was about selection control structure which consisted of three paths as shown in Table 5.1. The numbers are in the Table represents the derived test cases by the each subject. The derived test cases are grouped based on the paths covered from the *pre-test* experiment. For example, subject 1 derived total 6 test cases for question (1) where 2 test cases covered path 1 and 4 test cases for path 2 but did not cover any test cases for path 3.

Table 5.2 Number of test cases to cover each path by current method for question (1)

| Current method: Question (1) | | | |
|---|---|---|---|
| Subjects | Path 1(Test Cases) | Path 2 (Test Cases) | Path3 (Test Cases) |
| 1 | 2 | 4 | 0 |
| 2 | 3 | 2 | 2 |
| 3 | 1 | 2 | 0 |
| 4 | 3 | 4 | 3 |
| 5 | 2 | 2 | 2 |
| 6 | 3 | 3 | 2 |
| 7 | 4 | 2 | 4 |
| 8 | 4 | 4 | 2 |
| 9 | 1 | 1 | 4 |
| 10 | 1 | 0 | 4 |

The following Table 5.3 shows the total number of test cases for current method derived by each subject for question (1).

Table 5.3 Total number of test cases by current method for question (1)

| Subjects | Total Test Cases (Question 1) |
|---|---|
| 1 | 6 |
| 2 | 7 |
| 3 | 3 |
| 4 | 10 |
| 5 | 6 |
| 6 | 8 |
| 7 | 10 |
| 8 | 10 |
| 9 | 6 |
| 10 | 5 |

### 5.1.2 Question (1): DyStruc-TDG Method

By using DyStruc-TDG method in *post-test* experiment the Table 5.4 shows the number of test cases to cover 3 paths mentioned in Table 5.1 by each subject. In this case the DyStruc-TDG method has generated test cases in a consistent way for each path and it considered each option individually based on MC/DC coverage concept. For example, path 1 (age>=21&& age <=200) has two options (age>=21) and (age <=200). According to MC/DC formula provided in **Chapter 2** (see **Table 2.7**) the DyStruc-TDG method has generated 3 test cases. Thus, another 2 paths (path 2 and path 3) also have 2 options. That is why the DyStruc-TDG method also has generated 3 test cases for each path.

Table 5.4 Number of test cases to cover by DyStruc-TDG method for question (1)

| DyStruc-TDG Method: Question (1) | | | |
|---|---|---|---|
| Subjects | Path 1(Test Cases) | Path 2 (Test Cases) | Path3 (Test Cases) |
| 1 | 3 | 3 | 3 |
| 2 | 3 | 3 | 3 |
| 3 | 3 | 3 | 3 |
| 4 | 3 | 3 | 3 |
| 5 | 3 | 3 | 3 |
| 6 | 3 | 3 | 3 |
| 7 | 3 | 3 | 3 |
| 8 | 3 | 3 | 3 |
| 9 | 3 | 3 | 3 |
| 10 | 3 | 3 | 3 |

The following Table 5.5 shows the total number of test cases is 9 which are generated by DyStruc-TDG method for question (1).

Table 5.5 Total number of test cases by DyStruc-TDG for question (1)

| Subjects | Total Test Cases (Question 1) |
|---|---|
| 1 | 9 |
| 2 | 9 |
| 3 | 9 |
| 4 | 9 |
| 5 | 9 |
| 6 | 9 |
| 7 | 9 |
| 8 | 9 |
| 9 | 9 |
| 10 | 9 |

The following line graph from Figure 5.2 shows the number of test cases coverage for current method and DyStruc-TDG method. The DyStruc-TDG method generated test cases in a consistent way and covered all the paths. For MC/DC coverage the DyStruc-TDG method considered each option individually. On the other hand, the subjects of current method derived test cases in an inconsistent way. Regarding the path coverage some subjects did not cover all the paths (see **Table 5.1**). Based on their deriving test cases they did not consider each option individually for MC/DC coverage.

From the line graph the subjects (4, 7, and 8) derived one extra test case than DyStruc-TDG method. In this case DyStruc-TDG method reduces one test case. On the other hand, the other subjects derived less test cases than DyStruc-TDG method. In this case, although the current method reduces number of test cases but did not cover all the paths and each option individually for selection control structures. Thus, current method did not derive consistent test cases to provide thoroughness testing where DyStruc-TDG method covered all the paths and options individually and provides the thoroughness of testing.

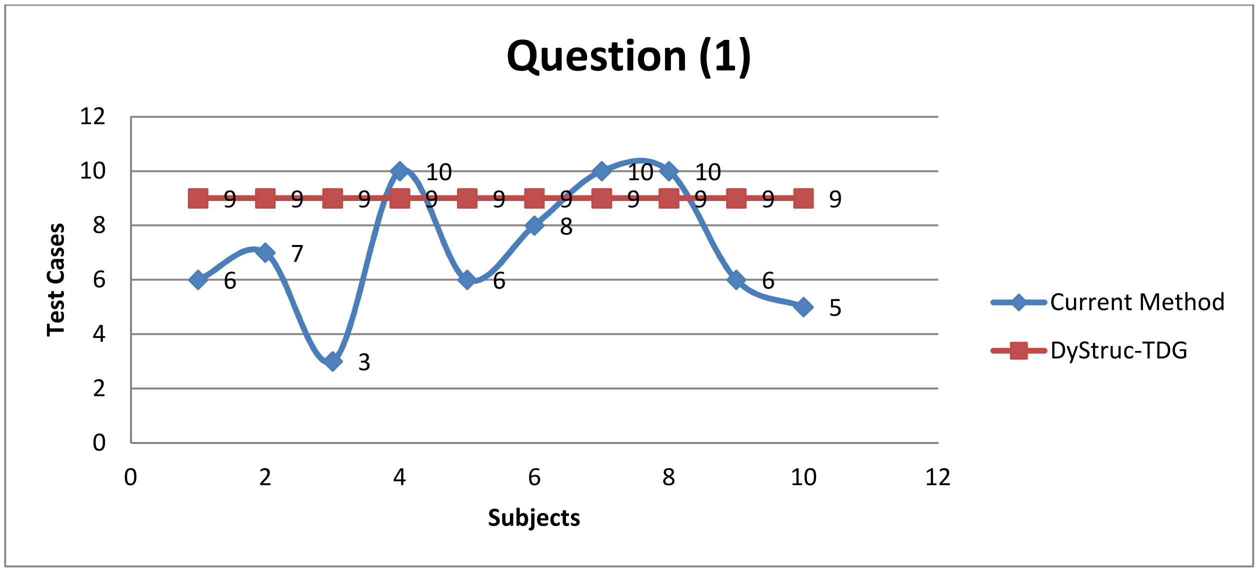
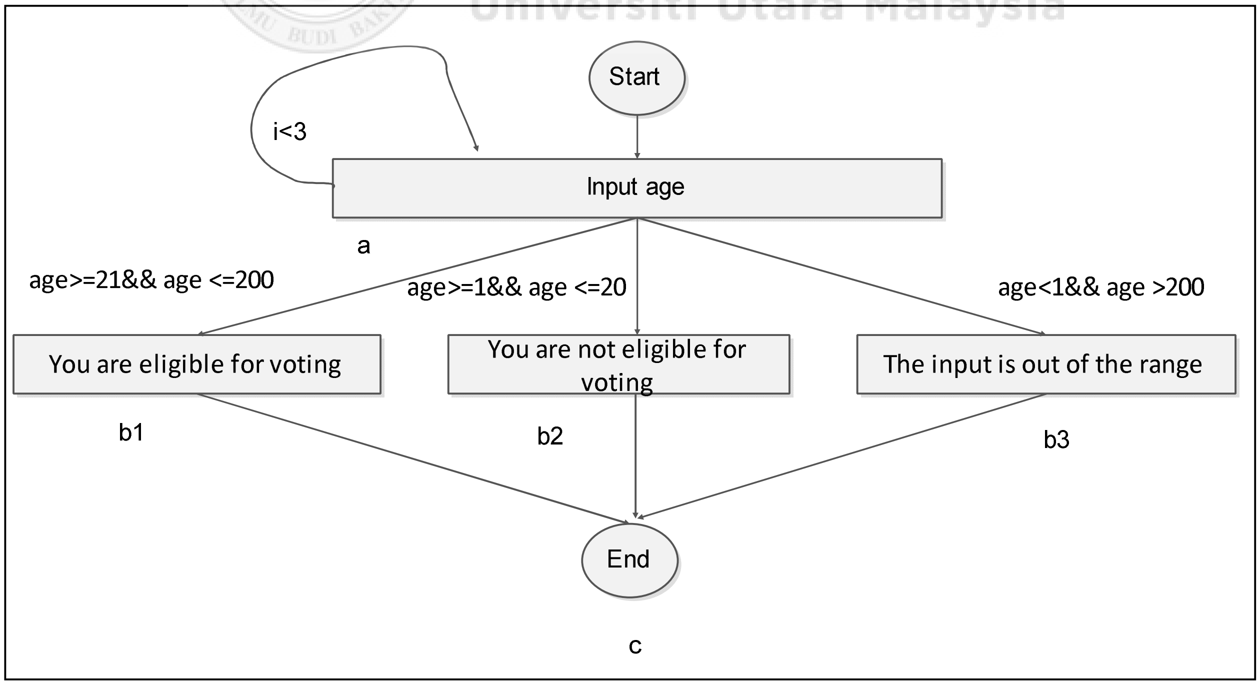


Figure 5.2 Test cases coverage between Current Method and DyStruc-TDG for question (1)

### 5.1.3 Question (2): Current method

The following Figure 5.3 shows the control flow graph of the repetition control structure for Question (2).



Start
i<3
Input age
a
age>=21&& age <=200
age>=1&& age <=20
age<1&& age >200
You are eligible for voting
You are not eligible for voting
The input is out of the range
b1
b2
b3
End
c

Figure 5.3 Control flow graph for repetition (counter loop)

The Table 5.6 shows the number of test cases for repetition control structure to cover 3 paths (path 1, path 2 and path 3) mentioned in Table 5.1 by each subject. In this case the DyStruc-TDG method has generated test cases in a consistent way for each path and it considered each option individually based on MC/DC coverage concept. For example, path 1 (age>=21&& age <=200) has two options (age>=21) and (age <=200). According to MC/DC formula in **Chapter 2** (see **Table 2.7**) the DyStruc-TDG method has generated 3 test cases. Thus, another 2 paths (path 2 and path 3) also have 2 options. That is why the DyStruc-TDG method has generated 3 test cases for each path.

Table 5.6 Number of test cases to cover by current method for question (2)

| Current method: Question (2) | | | |
|---|---|---|---|
| Subjects | Path 1 (Test Cases) | Path 2 (Test Cases) | Path3 (Test Cases) |
| 1 | 3 | 3 | 0 |
| 2 | 2 | 1 | 0 |
| 3 | 2 | 1 | 0 |
| 4 | 2 | 2 | 3 |
| 5 | 2 | 2 | 1 |
| 6 | 3 | 3 | 2 |
| 7 | 3 | 2 | 3 |
| 8 | 4 | 6 | 3 |
| 9 | 2 | 2 | 6 |
| 10 | 1 | 0 | 2 |

The following Table 5.7 shows the total number of test cases for current method derived by each subject for question (2).

Table 5.7 Total number of test cases by current method for question (2)

| Subjects | Total Test Cases (Question 2) |
|---|---|
| 1 | 6 |
| 2 | 3 |
| 3 | 3 |
| 4 | 7 |
| 5 | 5 |
| 6 | 8 |
| 7 | 8 |
| 8 | 13 |
| 9 | 10 |
| 10 | 3 |

### 5.1.4 Question (2): DyStruc-TDG Method

Based on *post-test* experiment the Table 5.8 shows the number of test cases to cover 3 paths mentioned in Table 5.1 by each subject. In this case the DyStruc-TDG method has generated test cases in a consistent way for each path and it considered each option individually based on MC/DC coverage concept. The example is same as provided for question (1).

Table 5.8 Number of test cases to cover by DyStruc-TDG method for question (2)

| DyStruc-TDG Method: Question (2) | | | |
|---|---|---|---|
| Subjects | Path 1 | Path 2 | Path3 |
| 1 | 3 | 3 | 3 |
| 2 | 3 | 3 | 3 |
| 3 | 3 | 3 | 3 |

| Subjects | Path 1 | Path 2 | Path3 |
|---|---|---|---|
| 4 | 3 | 3 | 3 |
| 5 | 3 | 3 | 3 |
| 6 | 3 | 3 | 3 |
| 7 | 3 | 3 | 3 |
| 8 | 3 | 3 | 3 |
| 9 | 3 | 3 | 3 |
| 10 | 3 | 3 | 3 |

The following Table 5.9 shows the total number of test cases is 9 which are generated by DyStruc-TDG method for question (2).

Table 5.9 Total number of test cases by DyStruc-TDG for question (2)

| Subjects | Total Test Cases (Question 2) |
|---|---|
| 1 | 9 |
| 2 | 9 |
| 3 | 9 |
| 4 | 9 |
| 5 | 9 |
| 6 | 9 |
| 7 | 9 |
| 8 | 9 |
| 9 | 9 |
| 10 | 9 |

From Figure 5.4, the following line graph shows the number of test cases coverage for current method and DyStruc-TDG method. The DyStruc-TDG method generated test cases in a consistent way and covered all the paths. For MC/DC coverage the DyStruc-TDG method considered each option individually. On the other hand, the subjects of current method derived test cases in an inconsistent way. Regarding the path coverage some subjects did not cover all the paths (see **Table 5.1**). Based on their deriving test cases they did not consider each option individually for MC/DC coverage.

From the line graph the subjects (8 and 9) derived four and one extra test cases respectively than DyStruc-TDG method. In this case DyStruc-TDG method reduces test cases than current method. On the other hand, the other subjects derived less test cases than DyStruc-TDG method. In this case, although the current method reduces number of test cases but did not cover all the paths and each option individually for selection control structures. Thus, current method did not derive consistent test cases to provide thoroughness testing where DyStruc-TDG method covered all the paths and options individually and provides the thoroughness of testing.

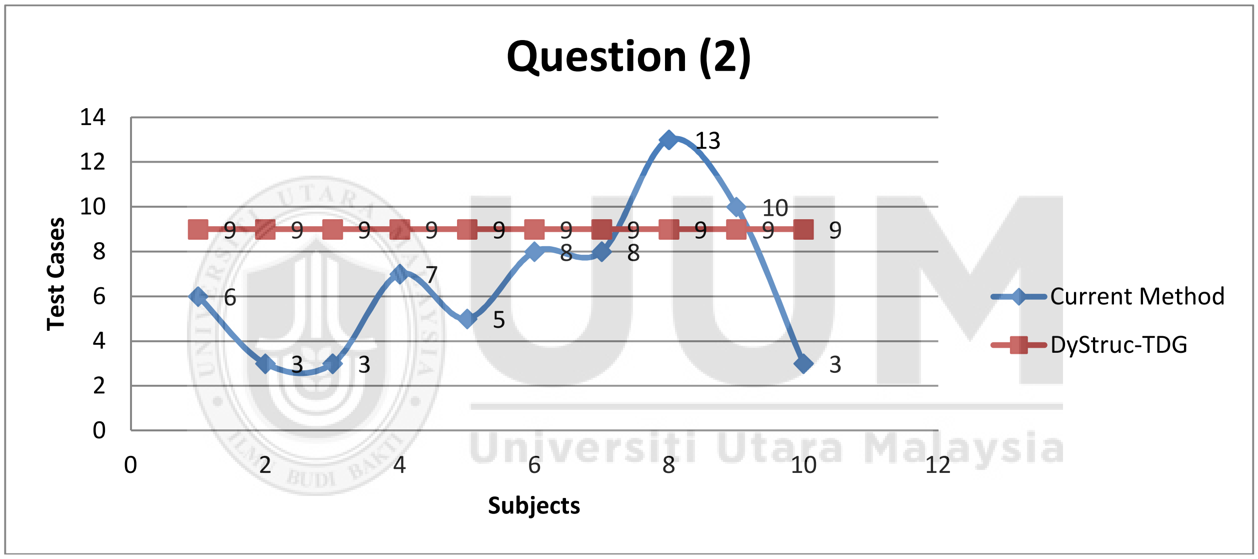


Figure 5.4 Test cases coverage between Current Method and DyStruc-TDG for question (2)

### 5.1.5 Question (3): Current method

The following Figure 5.5 shows the control flow graph of the repetition control structure for sentinel loop for Question (3).

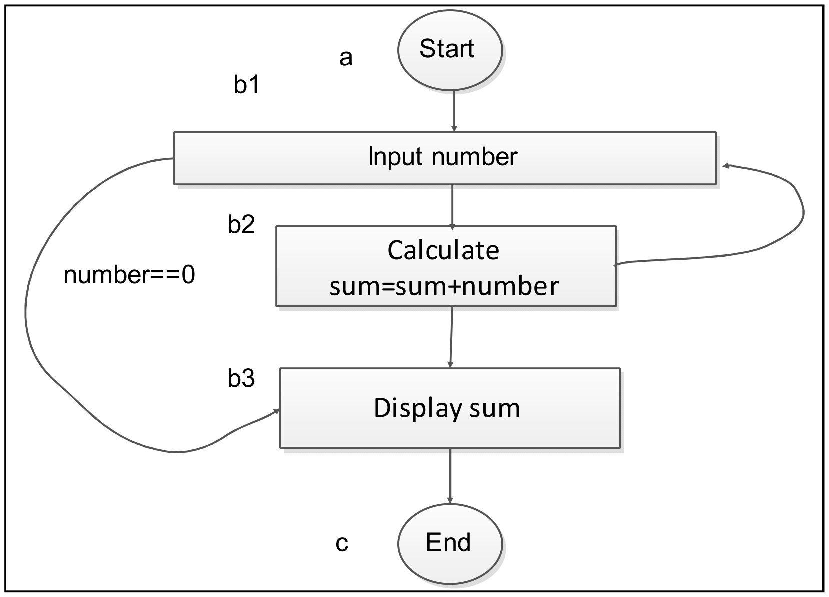


Figure 5.5 Control flow graph for repetition (sentinel loop)

Based on conducted *pre-test* experiment, the following Table 5.10 shows the number of derived test cases for current method. The question was about repetition control structure for sentinel loop which consisted of one path (number = =0). The numbers are in the Table represents the derived test cases by the each subject. The derived test cases are grouped based on the path covered from the *pre-test* experiment. For example, subject 1 derived total (4) test cases for question (3) where 4 test cases covered path 1.

Table 5.10 Number of test cases to cover by current method for question (3)

| Current method: Question (3) | |
|---|---|
| Subjects | Path 1(Test Cases) |
| 1 | 4 |
| 2 | 3 |
| 3 | 4 |
| 4 | 6 |
| 5 | 3 |
| 6 | 5 |
| 7 | 4 |

| Subjects | Path 1(Test Cases) |
|---|---|
| 8 | 6 |
| 9 | 3 |
| 10 | 3 |

### 5.1.6 Question (3): DyStruc-TDG Method

By using DyStruc-TDG method in *post-test* experiment the Table 5.11 shows the number of test cases to cover 1 path (number = =0). In this case the DyStruc-TDG method has generated test cases in a consistent way for path 1 and it considered one option individually based on MC/DC coverage concept.

For example, path 1 (number = =0) has one option. Based on MC/DC formula in **Chapter 2** (see **Table 2.7**) the DyStruc-TDG method has generated 2 test cases for one option.

Table 5.11 Total number of test cases by DyStruc-TDG method for question (3)

| DyStruc-TDG: Question (3) | |
|---|---|
| Subjects | Path 1 (Test Cases) |
| 1 | 2 |
| 2 | 2 |
| 3 | 2 |
| 4 | 2 |
| 5 | 2 |
| 6 | 2 |
| 7 | 2 |
| 8 | 2 |
| 9 | 2 |
| 10 | 2 |

The line graph generated in Figure 5.6 for question (3) shows the derived test cases by the current method and DyStruc-TDG method. In this case the current method line is above than DyStruc-TDG method. The reason of producing the line above is to deriving more test cases than DyStruc-TDG method generated. The question is regarding sentinel loop where this type of loop consists of two values either true or false. In current method, the subjects derived more test cases for false value where only 1 value is sufficient to cover adequate testing.

Table 5.12 Test data for sentinel loop in current method

| True value | False value | |
|---|---|---|
| 0 | 1 | |
| | -1 | Redundant test cases |
| | 1000 | |
| | 20 | |
| | A | |

For instance, based on above Table 5.12 in current method subject (4) derived total 6 test cases for true and false value where 1 test case is for true value and another 5 test cases for false value. Thus, subject (6) derived 4 redundant test cases for false value where only 1 test case is enough for false value regardless of any data type. On the other hand, the DyStruc-TDG method generates only 2 test cases (see **Table 5.11**) for true and false value which is adequate for thoroughness testing in path coverage and MC/DC coverage.

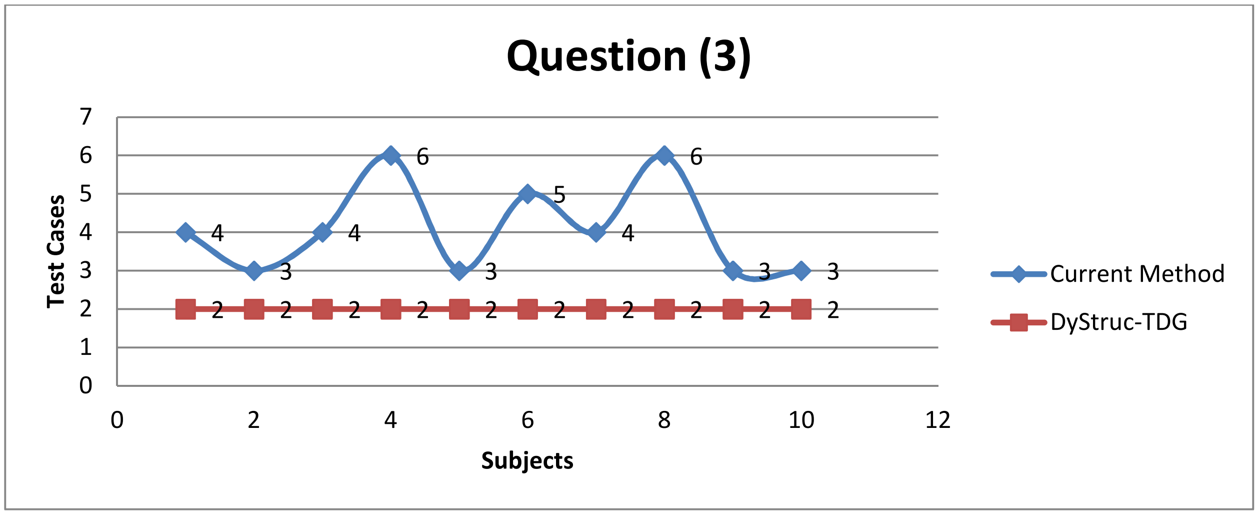


Figure 5.6 Test cases coverage between Current Method and DyStruc-TDG for question (3)

### 5.1.7 Question (4): Current method

The following Figure 5.7 shows the control flow graph of repetition and selection control structures for Question (4).

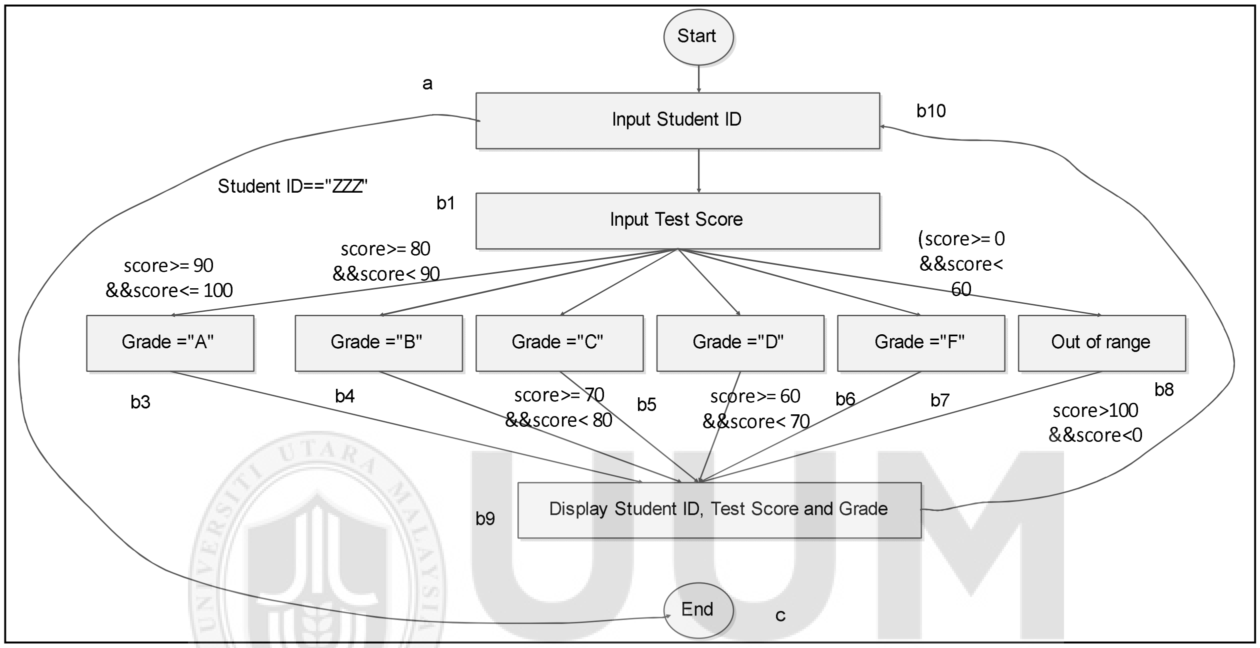


Figure 5.7 Control flow graph for repetition and selection control structures

The Table 5.13 shows the number of paths for selection and repetition control structures which consisted of 7 paths (path 1, path 2, path 3, path 4, path 5, path 6 and path 7) and Table 5.9 shows the number of test cases derived by ten subjects to cover the selection control structures by using current method.

Table 5.13 Number of paths for selection and repetition control structures

| Number of Paths | Path Condition |
|---|---|
| Path1 | (!studentID.equalsIgnoreCase("zzz")) |
| Path 2 | (testScore>= 90 && testScore<= 100) |
| Path 3 | testScore>= 80 && testScore< 90) |
| Path 4 | (testScore>= 70 && testScore< 80) |
| Path 5 | (testScore>= 60 && testScore< 70) |

| Number of Paths | Path Condition |
|---|---|
| Path 6 | (testScore>= 0 && testScore< 60) |
| Path 7 | (testScore>100 && testScore<0) |

Based on conducted *pre-test* experiment, the following Table 5.14 shows the number of derived test cases for current method. The question was about repetition and selection control structure which consisted of seven paths as shown in Table 5.12. The numbers are in the Table represents the derived test cases by the each subject. The derived test cases are grouped based on the paths covered from the *pre-test* experiment. For example, subject 1 derived total 13 test cases for question (4) where 7 test cases covered path 1, 1 test case for path (2, 3, 6 and 7), 2 test cases covered path 4 and 0 test case for path 5.

Table 5.14 Number of test cases to cover by current method for question (4)

| Current method: Question (4) | | | | | | | |
|---|---|---|---|---|---|---|---|
| Subjects | Path1(TC) | Path2(TC) | Path3(TC) | Path4(TC) | Path5(TC) | Path6(TC) | Path7(TC) |
| 1 | 7 | 1 | 1 | 2 | 0 | 1 | 1 |
| 2 | 0 | 3 | 3 | 3 | 3 | 3 | 2 |
| 3 | 5 | 0 | 1 | 0 | 1 | 1 | 0 |
| 4 | 11 | 1 | 0 | 1 | 1 | 3 | 2 |
| 5 | 0 | 0 | 2 | 1 | 0 | 0 | 1 |
| 6 | 5 | 3 | 2 | 2 | 2 | 2 | 1 |
| 7 | 2 | 4 | 4 | 2 | 5 | 2 | 2 |
| 8 | 18 | 4 | 2 | 3 | 4 | 2 | 0 |
| 9 | 7 | 1 | 1 | 1 | 1 | 1 | 3 |
| 10 | 9 | 2 | 0 | 1 | 1 | 1 | 3 |

The following Table 5.15 shows the total number of test cases for current method derived by each subject for question (4).

Table 5.15 Total number of test cases by current method for question (4)

| Subjects | Total Test Cases (Question 4) |
|---|---|
| 1 | 13 |
| 2 | 17 |
| 3 | 8 |
| 4 | 19 |
| 5 | 4 |
| 6 | 17 |
| 7 | 21 |
| 8 | 33 |
| 9 | 15 |
| 10 | 17 |

### 5.1.8 Question (4): DyStruc-TDGMethod

By using DyStruc-TDG method in *post-test* experiment the Table 5.16 shows the number of test cases to cover 7 paths mentioned in Table 5.13 by each subject. In this case the DyStruc-TDG method has generated test cases in a consistent way for each path and it considered each option individually based on MC/DC coverage concept.

For example, path 1 (!studentID.equalsIgnoreCase("zzz")) has one option. By applying MC/DC formula from **Chapter 2** (see **Table 2.7**) the DyStruc-TDG method has generated 2 test cases for one option. But, another 6 paths (path 2, path 3, path 4, path 5, path 6 and path 7) have 2 options. That is why the DyStruc-TDG method has generated 3 test cases for each path.

Table 5.16 Number of test cases to cover by DyStruc-TDG method for question (4)

| DyStruc-TDG Method: Question (4) | | | | | | | |
|---|---|---|---|---|---|---|---|
| Subjects | Path1(TC) | Path2(TC) | Path3(TC) | Path4(TC) | Path5(TC) | Path6(TC) | Path7(TC) |
| 1 | 2 | 3 | 3 | 3 | 3 | 3 | 3 |
| 2 | 2 | 3 | 3 | 3 | 3 | 3 | 3 |
| 3 | 2 | 3 | 3 | 3 | 3 | 3 | 3 |
| 4 | 2 | 3 | 3 | 3 | 3 | 3 | 3 |
| 5 | 2 | 3 | 3 | 3 | 3 | 3 | 3 |
| 6 | 2 | 3 | 3 | 3 | 3 | 3 | 3 |
| 7 | 2 | 3 | 3 | 3 | 3 | 3 | 3 |
| 8 | 2 | 3 | 3 | 3 | 3 | 3 | 3 |
| 9 | 2 | 3 | 3 | 3 | 3 | 3 | 3 |
| 10 | 2 | 3 | 3 | 3 | 3 | 3 | 3 |

The following Table 5.17 shows the total number of test cases is 20 which are generated by DyStruc-TDG method for question (4).

Table 5.17 Total number of test cases by DyStruc-TDG method for question (4)

| Subjects | Total Test Cases (Question 4) |
|---|---|
| 1 | 20 |
| 2 | 20 |
| 3 | 20 |
| 4 | 20 |
| 5 | 20 |
| 6 | 20 |
| 7 | 20 |
| 8 | 20 |
| 9 | 20 |
| 10 | 20 |

From Figure 5.8 the following line graph shows the number of test cases coverage for current method and DyStruc-TDG method. The DyStruc-TDG method generated test cases in a consistent way and covered all the paths. For MC/DC coverage the DyStruc-TDG method considered each option individually. On the other hand, the subjects of current method derived test cases in an inconsistent way. Regarding the path coverage some subjects did not cover all the paths (see **Table 5.1**). Based on their deriving test cases they did not consider each option individually for MC/DC coverage.

From the line graph the subjects (7 and 8) derived one and thirteen extra test cases respectively than DyStruc-TDG method. In this case DyStruc-TDG method reduces one and thirteen test cases. On the other hand, the other subjects derived less test cases than DyStruc-TDG method. In this case, although the current method reduces number of test cases but did not cover all the paths and each option individually for selection control structures. Thus, current method did not derive consistent test cases to provide thoroughness testing where DyStruc-TDG method covered all the paths and options individually and provides the thoroughness of testing.

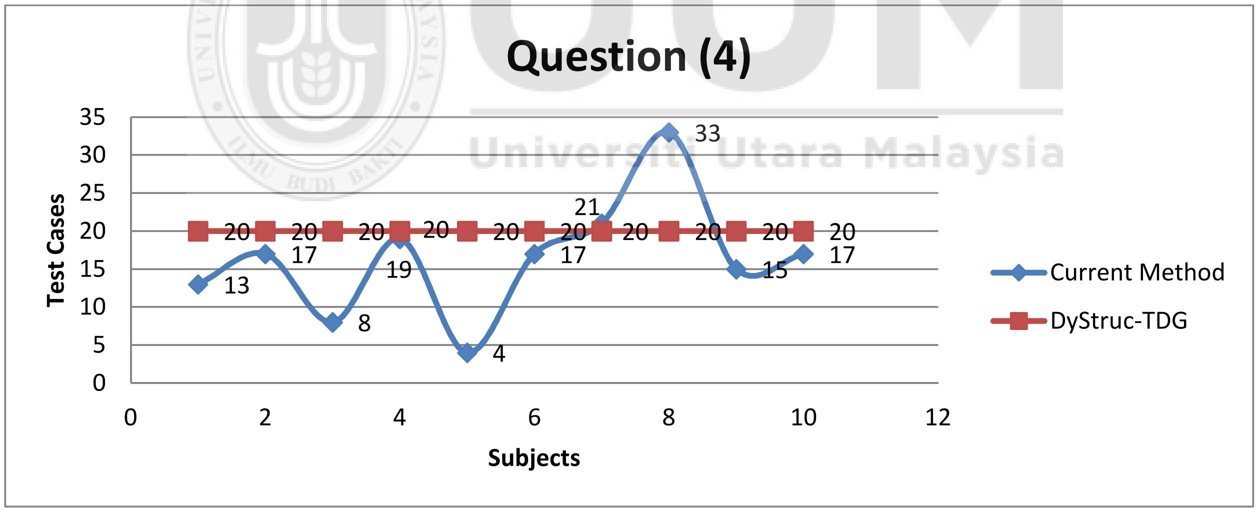
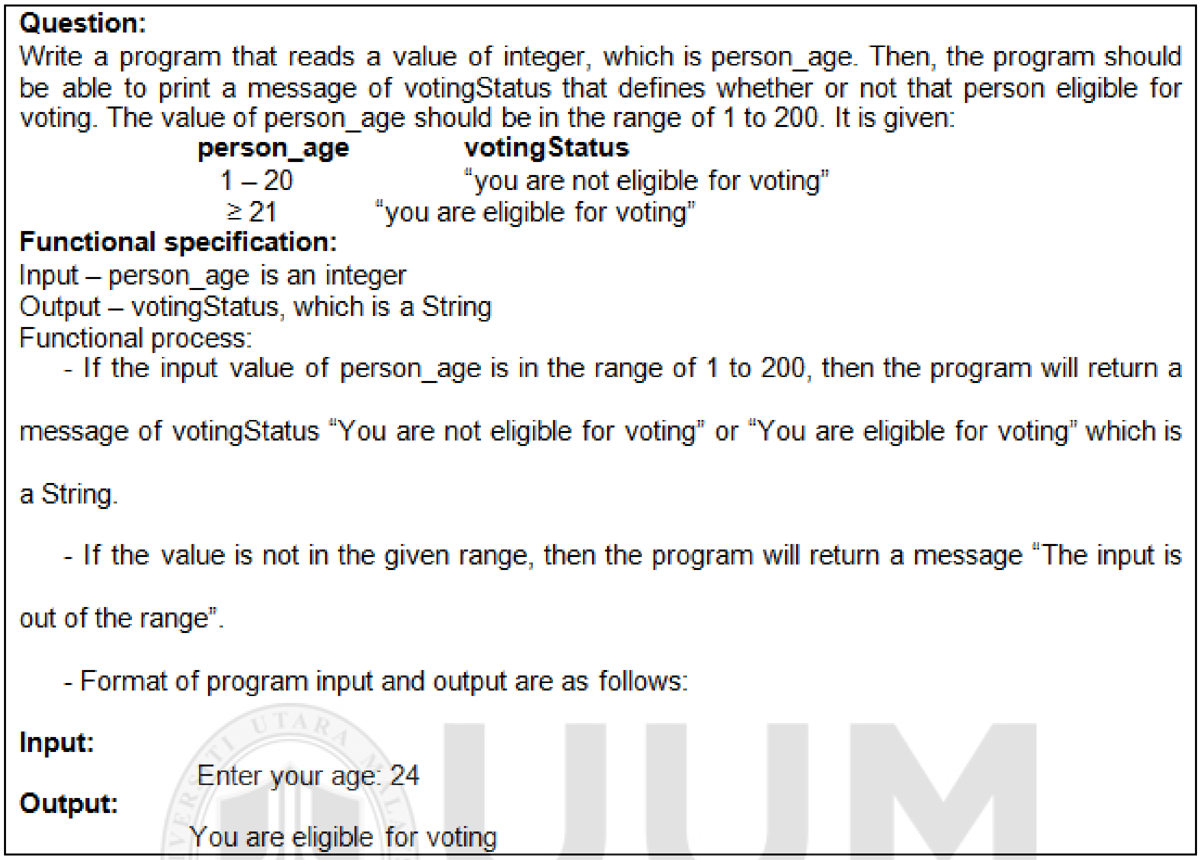


Figure 5.8 Test cases coverage between Current Method and DyStruc-TDG for question (4)

## 5.2 Comparative Evaluation

Qualitative evaluation also presented in this study to compare in terms of test data adequacy for structural testing. The comparison was done among three studies in structural testing namely Ihantola (2006) and Tillmann *et al*., (2013); Rohaida (2014) and DyStruc-TDG. A sample of programming exercise is used for this comparison as shown in Figure 5.9.



**Question:**
Write a program that reads a value of integer, which is person_age. Then, the program should be able to print a message of votingStatus that defines whether or not that person eligible for voting. The value of person_age should be in the range of 1 to 200. It is given:

| **person_age** | **votingStatus** |
|---|---|
| 1 – 20 | "you are not eligible for voting" |
| ≥ 21 | "you are eligible for voting" |

**Functional specification:**
Input – person_age is an integer
Output – votingStatus, which is a String
Functional process:
- If the input value of person_age is in the range of 1 to 200, then the program will return a message of votingStatus "You are not eligible for voting" or "You are eligible for voting" which is a String.
- If the value is not in the given range, then the program will return a message "The input is out of the range".
- Format of program input and output are as follows:

**Input:**
```
Enter your age: 24
```
**Output:**
```
You are eligible for voting
```

Figure 5.9 Sample of programming exercise

The following Table 5.18 shows the comparison of three studies for structural testing.

Table 5.18 Comparison of three studies

| Criteria of comparison | Ihantola (2006) and Tillmann *et al*., (2013) | Rohaida (2014) | DyStruc-TDG |
|---|---|---|---|
| Number of test cases | 4 | 5 | 9 |
| Test data coverage | 1. age>=21&& age <=200<br>2. age>=1&& age <=20<br>3. age<1<br>4. age >200 | 1. age>=21&& age <=200<br>2. age>=1&& age <=20<br>3. age<1<br>4. age >200<br>5. Illegal path condition | 1. age>=21&& age <=200<br>2. age>=1&& age <=20<br>3. age<1<br>4. age >200 |
| Values of test data | Input parameter based on path condition | Lecturer need to assign test data | Automated generated |

Based on the comparison it shows that, Ihantola (2006) and Tillmann *et al*., (2013) and Rohaida (2014) have derived respectively 4 and 5 test cases. On the other hand, DyStruc-TDG method has derived 9 test cases. However, in terms of coverage structural testing DyStruc-TDG method has used MC/DC coverage concept. DyStruc-TDG method has covered each option individually as part of Boolean expression where it considered true and false value for each option individually. In terms of performing structural testing proposed by Rohaida (2014) it is required human involvement to assign test data where DyStruc-TDG methods generates test data automatically.Thus, it concludes that DyStruc-TDG method has covered more thorough testing than another two studies which are proposed by Tillmann *et al*., (2013) and Rohiada (2014) in terms of path coverage and MC/DC coverage.

## 5.3 Summary

In summary, this chapter has explained the evaluation process into two parts namely controlled experiment and qualitative evaluation. In controlled experiment has discussed about pre-test and post-test experiment. The number of test cases derived through current method and DyStruc-TDG method by the ten subjects was presented. Figure 5.5 to Figure 5.14 described the findings of derived test cases in order to compare between current method and DyStruc-TDG method. Finally, the second part involves the qualitative evaluation where two studies were presented in order to compare with DyStruc-TDG by providing a sample of programming exercises.

# CHAPTER 6

# CONCLUSION

This study is with regard proposing a test data generation method for deriving and generating an adequate set of test data to perform the dynamic structural testing in automatic programming assessment (APA). Previous studies found some gaps and lacking of adequate coverage of test data in testing of student's program. Furthermore, the previous studies have presented many methods in terms generating test data that fulfill some coverage of software testing criteria but very few studies have integrated both in the context of APA. By having a test data generator to represent the proposed method (DyStruc-TDG), it is able to assist lecturers of introductory programming courses to prepare and generate adequate set of test data automatically regardless of having the optimal knowledge in designing of test cases.

In order to formulating research problem and deciding the solution to the problem, this study conducted a thorough literature review. Their findings have described in details in **Chapter 2**. It covers the existing methods of test data generation in the context of software testing area and integration of test data generation with APA. The next sections will discuss each of provided solutions in order to answer research questions, contributions as well as overall conclusion obtained within this study.

## 6.1 Revisit of Research Questions and Objectives

This section summarizes the identified research questions and research objectives which were formulated in **Chapter 1**.

Based on literature review, this study identified the problem and formulated the main research question as: "***How to generate a set of test data that do satisfy the test adequacy criteria to adhere the coverage of structural testing of a program executed for APA?***"

Based on the above main research question, this study was conducted the following specific sub questions:

1. How to construct a test data generation method to achieve the means of deriving an adequacy set of the test data for dynamic structural testing in APA?(***RQ-1***)

2. How to measure the adequacy of test data as derived in (1) in the context of APA? (***RQ-2***)

Based on research questions above, this study aims to achieve the following objectives:

1. To construct a test set which include an adequate of test data to represent DyStruc-TDG (***RO-1***)

2. To measure the adequacy of test data derived from DyStruc-TDG in the context of programming assessment(***RO-2***)

The next section will discuss the solutions in order to answer the provided research questions as well as the overall conclusion obtained within this study.

### 6.1.1 Discussion on Research Question (1)

In order to answer this question, the related literature was reviewed where it's mainly focused on software testing adequate coverage criteria. In order to know how adequate the testing is, it is required to test the coverage of the program. Based on the literature review, this study found the ranking of code coverage which helps to derive an adequacy set of test data for dynamic structural testing in APA. This study has adapted two test data adequacy coverage criteria which are path and Modified Condition/Decision Coverage.

The objective of the path testing is to ensure that every path in the program travelled through programs executed at least once. In structural test data generation, test cases are derived in such a way that every path is executed at least once as path coverage. It ensures coverage of all the paths from start to end.

Next, the Modified Condition/Decision Coverage is one of the structural coverage criteria that is used to assist in the assessment of adequacy of test data generation. This coverage criterion explains that every condition in the decision independently affects the decision's outcome. MC/DC coverage criterion reduces the number of test cases and generates adequate test data.

In overall, by adopting the path coverage and Modified Condition/Decision Coverage test adequacy coverage criteria the DyStruc-TDG method has achieved the means of deriving an adequacy set of test data for dynamic structural testing in APA.

### 6.1.2 Discussion on Research Question (2)

Evaluation process measures the test data generation method whether or not it improves the completeness coverage of test data adequacy criteria of reliability issue. The evaluation with regard to this aim is purposely to answer the second research question (**RQ-2**). To measure the adequacy of test data a controlled experiment and qualitative evaluation has been carried out for this study.

After constructing the method, it is important to test the method in order to measure the adequacy of test data. Therefore, a prototype is developed. A controlled experiment is conducted to evaluate the developed method called DyStruc-TDG. The controlled experiment consists of *pre-test* and *post-test* experiment. By using current method in *pre-test* experiment subjects derived test data for each question. Then, in *post-test* experiment DyStruc-TDG method was used to generate test data for the same questions.

In *pre-test* experiment the numbers of deriving test cases by the subjects are not consistent and did not cover all the paths. Some subjects derived more test cases than necessary which causes more workload. On the other hand, DyStruc-TDG method generates adequate set of test cases in a consistent way and covered all the paths and each option individually.

Qualitative evaluation also presented in this study to compare in terms of test data adequacy for structural testing. The comparison of DyStruc-TDG was done among three studies in structural testing. From the comparison of three studies it concludes that DyStruc-TDG method covers more thorough testing and generates adequate set of test data for APA.

In conclusion, based on controlled experiment and qualitative evaluation the DyStruc-TDG method generates adequate set of test data for path coverage and MC/DC coverage which covers the thoroughness of testing.

## 6.2 Contribution of the Study

This study is an attempt to adapt existing test adequacy criteria applied in software testing field as a test data generation method to perform dynamic structural testing for automatic programming assessment. This study contributes in terms of theoretical and practical context for APA. In theoretical perspective, this study enhances the existing researches in APA by providing the means of deriving and generating test data automatically by integrating the path and MC/DC coverage. These two structural codes coverage provide a significant impact in reducing the number of test cases required to test students' programming solutions in terms of the aspect of structural testing. In addition, a more thorough testing aspect is considered as each individual condition has been a part of the means of deriving the test cases.
On the other hand, in practical perspective, this study contributes a physical deliverable that is a test data generator to provide a medium of generating test data automatically for the usage of lecturers. Through this generator, it is able to assist the lecturers who teach programming courses to generate test data and test cases to perform automatic programming assessment regardless of having a particular knowledge of test cases design. Besides, indirectly the lecturers' workload can be reduced effectively since the typical manual assessments are always prone to errors.

## 6.3 Limitations and Recommendations

The limitations of this study and possible future recommendations are as follow:

### (i) Negative Testing

This study does not cover the negative testing criterion in order to derive and generate test data. Negative testing criterion provides the invalid test data as input into the program. This is to test the application "does not do anything that it is not supposed to do". By including the negative testing, the ideal test criterion can be achieved. Thus, more through testing can be included. The main concerned of excluding this criterion because the main concern of this study is to focus on each individual condition as part of the Boolean expression consisted in each path to cover MC/DC coverage. It is just enough to cover only true and false values for each condition.

However, in future recommendation, by including negative testing criterion it will resolve the limitation in terms of achieving ideal test criterion.

**(ii) Only Integer Value for Counter Loop**

As a prototype was developed to generate test data for automatic programming assessment, there is an implementation limitation of this prototype. In counter loop the value for comparison variable must be an integer instead of assigning any referring variable. For example the counter loop is: ***for (int i=0;i<5;i++).*** In this case, the value for comparison variable (i<5) must be an integer which is 5. In this prototype any referring variable cannot be assigned such as (i<number) where number is a referring a variable.

The limitation of this implementation can be resolved by providing in string data type for counter loop. Thus, it will resolve the limitation of assigning string variable.

## 6.4 Conclusion

Automatic programming assessment plays a vital role in academic in order to assess students program and marking their grade. At the same time, APA is assisting lecturers by reducing their workload. In order to testing students program it is required to generate set of test data. Therefore, in APA the generation of test data is very important. In software testing area, there are many automated test data generation methods are available but very few studies on APA used it. Integration of both test data generation and APA has only limited studies that provide testing coverage for quality program.

Therefore, this study proposed a method to derive and generate an adequate set of test data to perform structural testing in APA or so called DyStruc-TDG. This method can assist lecturers to generate an adequate set of test data for student's program. This method reduces the workload and time of lecturers as well as it assists them generating adequate test cases regardless of having optimal expertise in the knowledge of designing of test cases.

# REFERENCES

Bertolino, A., & Marchetti, E. (2005). A brief essay on software testing.*Software Engineering, 3rd edn. Development process*, *1*, 393-411.

Blumenstein, M., Green, S., Nguyen, a., & Muthukkumarasamy, V. (2004). GAME: a Generic Automated Marking Environment for programming assessment. International Conference on Information Technology: Coding and Computing, 2004. Proceedings. ITCC 2004., 212–216 Vol.1.

Burnstein, I. (2003). *Practical Software Testing*, Springer-Verlag, New York.

Cheng, Z., Monahan, R., & Mooney, A. (2011). nExaminer: A semi-automated computer programming assignment assessment framework for Moodle.

Choy, M., Nazir, U., Poon, C. K., & Yu, Y. T. (2005). Experiences in using an automated system for improving students' learning of computer programming. In *Advances in Web-Based Learning–ICWL 2005* (pp. 267-272). Springer Berlin Heidelberg.

Chu, H. D. (1997). *An evaluation scheme of software testing techniques* (pp. 259-262). Springer US.

Clarke, L. a. (1976). A System to Generate Test Data and Symbolically Execute Programs. IEEE Transactions on Software Engineering, SE-2(3), 215–222.

Clarke, L. A. (1976). A system to generate test data and symbolically execute programs. *Software Engineering, IEEE Transactions on*, (3), 215-222.

Davis, F. D. (1989). Perceived usefulness, perceived ease of use, and user acceptance of information technology. *MIS quarterly*, 319-340.

Edvardsson, J. (1999, October). A survey on automatic test data generation. In *Proceedings of the 2nd Conference on Computer Science and Engineering* (pp. 21-28).

Edvardsson, J. (1999, October). A survey on automatic test data generation. In *Proceedings of the 2nd Conference on Computer Science and Engineering*(pp. 21-28).

Foong, O. M., Tran, Q. T., Yong, S. P., & Rais, H. M. (2014, June). Swarm inspired test case generation for online C++ programming assessment. In*Computer and Information Sciences (ICCOINS), 2014 International Conference on* (pp. 1-5). IEEE.

Fraenkel, J. R., & Wallen, N. E. (1993). *How to design and evaluate research in education* (Vol. 7). New York: McGraw-Hill.

Ghani, K., & Clark, J. A. (2009, September). Automatic test data generation for multiple condition and MCDC coverage. In *Software Engineering Advances, 2009. ICSEA'09. Fourth International Conference on* (pp. 152-157). IEEE.

Ghani, K., & Clark, J. A. (2009, September). Automatic test data generation for multiple condition and MCDC coverage. In *Software Engineering Advances, 2009. ICSEA'09. Fourth International Conference on* (pp. 152-157). IEEE.

Goodenough, J. B., & Gerhart, S. L. (1975). Toward a theory of test data selection. *Software Engineering, IEEE Transactions on*, (2), 156-173.

Guo, M., Chai, T., & Qian, K. (2010, April). Design of Online Runtime and Testing Environment for Instant Java Programming Assessment. In*Information Technology: New Generations (ITNG), 2010 Seventh International Conference on* (pp. 1102-1106). IEEE.

Gupta, N., Mathur, A. P., & Soffa, M. L. (1998). Automated test data generation using an iterative relaxation method. *ACM SIGSOFT Software Engineering Notes*, *23*(6), 231-244.

Gupta, S., & Dubey, S. K. (2012). Automatic Assessment of Programming assignment, 452003, 315–323.

Hakulinen, L., & Malmi, L. (2014, June). QR code programming tasks with automated assessment. In *Proceedings of the 2014 conference on Innovation & technology in computer science education* (pp. 177-182). ACM.

Hayhurst, K. J., Veerhusen, D. S., Chilenski, J. J., & Rierson, L. K. (2001). A practical tutorial on modified condition/decision coverage.

Ihantola, P. (2006). Automatic test data generation for programming exercises with symbolic execution and Java PathFinder. *Master's thesis, Helsinki University of Technology, Departement of Theoretical Computer Science*.

IPL Information Processing Ltd. (197a). Designing Unit Test Cases. Available http://www.ipl.com/pdf p0823.pdf. Retrieved on: 20 Feb 2009

Jackson, D. (2000). A semi-automated approach to online assessment. *ACM SIGCSE Bulletin*, *32*(3), 164-167.

Jackson, D., & Usher, M. (1997). Grading student programs using ASSYST. ACM SIGCSE Bulletin, 29(1), 335–339.

James, R., Ivar, J., & Grady, B. (1999). The unified modeling language reference manual. *Reading: Addison Wesley*.

Koomen, T., & Pol, M. (1999). *Test process improvement: a practical step-by-step guide to structured testing*. Addison-Wesley Longman Publishing Co., Inc.

Korel, B. (1990). Automated software test data generation. *Software Engineering, IEEE Transactions on*, *16*(8), 870-879.

Korel, B. (1996). Automated test data generation for programs with procedures. ACM SIGSOFT Software Engineering Notes, 21(3), 209–215.

Lahtinen, E., Ala-Mutka, K., & Järvinen, H. M. (2005, June). A study of the difficulties of novice programmers. In *ACM SIGCSE Bulletin* (Vol. 37, No. 3, pp. 14-18). ACM.

Latiu, G. I., Cret, O. A., & Vacariu, L. (2012). Automatic Test Data Generation for Software Path Testing Using Evolutionary Algorithms. 2012 Third International Conference on Emerging Intelligent Data and Web Technologies, 1–8.

Luck, M., & Joy, M. (1999). A secure on-line submission system. Software: Practice and Experience, 29(8), 721–740.

Malmi, L., Karavirta, V., Korhonen, A., Nikander, J., Seppälä, O., & Silvasti, P. (2004). Visual Algorithm Simulation Exercise System with Automatic Assessment: TRAKLA2. *Informatics in education*, *3*(2), 267-288.

McMinn, P. (2004). Search-based software test data generation: A survey.*Software Testing Verification and Reliability*, *14*(2), 105-156.

Monpratarnchai, S., Fujiwara, S., Katayama, A., & Uehara, T. (2014). Automated testing for Java programs using JPF-based test case generation.*ACM SIGSOFT Software Engineering Notes*, *39*(1), 1-5.

Offutt, J., Liu, S., Abdurazik, A., & Ammann, P. (2003). Generating test data from state-based specifications. Software Testing, Verification and Reliability, 13(1), 25–53.

Pargas, R. P., Harrold, M. J., & Peck, R. R. (1999). Test-data generation using genetic algorithms. *Software Testing Verification and Reliability*, *9*(4), 263-282.

Pressman, R. S. (2005). *Software engineering: a practitioner's approach*. Palgrave Macmillan.

Rayadurgam, S., & Heimdahl, M. P. E. (2003, December). Generating MC/DC Adequate Test Sequences Through Model Checking. In *SEW* (p. 91).

Romli, R. (2014). Test Data Generation Framework for Automatic Programming Assessment, 84–89.

Romli, R., Sulaiman, S., & Zamli, K. Z. (2010). Automatic Programming Assessment and Test Data Generation, 00(c).

Romli, R., Sulaiman, S., & Zamli, K. Z. (2013). Designing a Test Set for Structural Testing in Automatic Programming Assessment.

Rumbaugh, J. (2003). Object-oriented analysis and design (OOAD).

Saikkonen, R., Malmi, L., & Korhonen, A. (2001, June). Fully automatic assessment of programming exercises. In *ACM Sigcse Bulletin* (Vol. 33, No. 3, pp. 133-136). ACM.

Sekaran, U. (2006). *Research methods for business: A skill building approach*. John Wiley & Sons.

Tillmann, N., & Halleux, J. De. (2008). Pex – White Box Test Generation for . NET, 134–153.

Tillmann, N., De Halleux, J., Xie, T., Gulwani, S., & Bishop, J. (2013, May). Teaching and learning programming and software engineering via interactive gaming. In *Software Engineering (ICSE), 2013 35th International Conference on* (pp. 1117-1126). IEEE.

Truong, N., Bancroft, P., & Roe, P. (2005). Learning to program through the web. ACM SIGCSE Bulletin, 37(3), 9.

Varshney, S., & Mehrotra, M. (2013). Search based software test data generation for structural testing: a perspective. *ACM SIGSOFT Software Engineering Notes*, *38*(4), 1-6.

Venkatesh, V., Morris, M. G., Davis, G. B., & Davis, F. D. (2003). User acceptance of information technology: Toward a unified view. *MIS quarterly*, 425-478

Watkins, J., & Mills, S. (2010). *Testing IT: an off-the-shelf software testing process*. Cambridge University Press.

Zamli, K. Z., Ashidi, N., Isa, M., Fadel, M., & Klaib, J. (2007). A Tool for Automated Test Data Generation ( and Execution ) Based on Combinatorial Approach, 19–36.

Zhu, H., Hall, P. A., & May, J. H. (1997). Software unit test coverage and adequacy. *Acm computing surveys (csur)*, *29*(4), 366-427.

Zidoune, W., & Benouhiba, T. (2012). Targeted adequacy criteria for search-based test data generation.2012 International Conference on Information Technology and E-Services, 1-6.

## Appendix A: Experiment Assignments

**Question 1: Selection**

**Question:**
Write a program that reads a value of integer, which is person_age. Then, the program should be able to print a message of votingStatus that defines whether or not that person eligible for voting. The value of person_age should be in the range of 1 to 200. It is given:

| **person_age** | **votingStatus** |
|---|---|
| 1 – 20 | “you are not eligible for voting” |
| ≥ 21 | “you are eligible for voting” |

**Functional specification:**
Input – person_age is an integer
Output – votingStatus, which is a String
Functional process:
- If the input value of person_age is in the range of 1 to 200, then the program will return a message of votingStatus “You are not eligible for voting” or “You are eligible for voting” which is a String.
- If the value is not in the given range, then the program will return a message “The input is out of the range”.
- Format of program input and output are as follows:

**Input:**
```
Enter your age: 24
```
**Output:**
```
You are eligible for voting
```

```
public class TestVoting {
public static void main(String[] arg) {
    Scanner scan = new Scanner(System.in);
int age;
System.out.println("Enter your age:");
age = scan.nextInt();
if(age>=21&& age <=200)
System.out.println("You are eligible for voting");
elseif(age>=1&& age <=20)
System.out.println("You are not eligible for voting");
elseif(age<1&& age >200)
System.out.println("The input is out of the range");}
}
```

**Question 2: Repetition: Counter loop**

**Question:**
Modify **Question 1** so that it can process the same tasks for **THREE (3)**age values. The format of program input and output are as follows:

**Input and output:**
```
Person 1:
Enter your age: 24
You are eligible for voting
Person 2:
Enter your age: 19
You are not eligible for voting
Person 3:
Enter your age: 60
You are not eligible for voting
```

```
public class TestVoting {
public static void main(String[] arg) {
 Scanner scan = new Scanner(System.in);
int age;
for (inti = 0; i< 3; i++) {
System.out.println("Person " + (i+1));
System.out.print("Enter your age:");
age = scan.nextInt();
if(age>=21&& age <=200)
System.out.println("You are eligible for voting");
elseif(age>=1&& age <=20)
System.out.println("You are not eligible for voting");
elseif(age<1|| age >200)
System.out.println("The input is out of the range");
        }
    }
}
```

**Question 3: Repetition: Sentinel loop**

**Question:**
Write a program that reads and calculates the sum of an unspecified number of integers (data). The input 0 signifies the end of the input.
**Functional specification:**
Input – data, which is an integer data type
Output – sum, which is also an integer:

```
Input:
    7
Output:
    The sum is 7
```

Functional Process
- If data is an integer and is not 0, the program will keep on calculating sum.
- If data is an integer and is 0, the program will stop calculating the sum and return the sum value as the program output.

```
public class TestSentinel {

public static void main(String[] arg) {
 Scanner scan = new Scanner(System.in);
int number, sum = 0;
System.out.print("Enter a number:");
number = scan.nextInt();
while (number!= 0) {
sum = sum + number;
System.out.print("Enter a number:");
number = scan.nextInt();
        }
System.out.print("Sum is " + sum);
    }
```

## Question 4: Repetition + Selection

**Question:**
Write a Java program that reads student IDs and their test scores. The input “ZZZ” for student ID signifies the end of the input. For each student, the program outputs the student ID, test score and grade. The grade is defined based on the following conditions:

| test_score | grade |
|---|---|
| 90 ≤test_score≤ 100 | A |
| 80 ≤test_score< 90 | B |
| 70 ≤test_score< 80 | C |
| 60 ≤test_score< 70 | D |
| 0 ≤test_score< 60 | F |

The format of program input and output are as follow:

**Input and Output:**

```
S1111                                    →student ID
70                                       →test score
Student Id = S1111, test score = 70, and grade = D→ output
ZZZ
```

**Functional specification:**

```
public class TestSelectionSentinel {
public static void main(String[] arg) {
    Scanner scan = new Scanner(System.in);
    String studentID, grade="";
doubletestScore;
System.out.print("Enter your ID:");
studentID = scan.next();
while (!studentID.equalsIgnoreCase("zzz")) {
System.out.println();
System.out.print("Enter your test score:");
testScore = scan.nextDouble();
if (testScore>= 90 &&testScore<= 100) {
grade = "A";
      } else if (testScore>= 80 &&testScore< 90) {
grade = "B";
      } else if (testScore>= 70 &&testScore< 80) {
grade = "C";
      } else if (testScore>= 60 &&testScore< 70) {
grade = "D";
      } else if (testScore>= 0 &&testScore< 60) {
grade = "F";
      }
System.out.print("Student Id = " + studentID + ",test score = " + testScore + " and grade = " + grade);
System.out.print("Enter your ID:");
studentID = scan.next();   }}}
```

## Appendix B: Pre-Test and Post-Test Questions

Pusat Pengajian Pengkomputeran
School of Computing
Universiti Utara Malaysia

***Pre-test* Experiment– Test Data Adequacy of Dynamic Structural Test Data Generation for Programming Assessment**

Dear Evaluators,

This *pre-test* experiment intends to measure the degree of the current method used in preparing a set of test data to perform the dynamic structural testing in assessing students programming exercises. The current method means the way of preparing test data based on individual user's knowledge in a certain test cases design. All responses from this pre-test experiment are anonymous and will be strictly confidential.

Thank you for your kind co-operation.

Kind Regards,
Md. Shahadath Sarker (816283)
MSc. IT (Course Work and Dissertation)

Supervisor:
Rohaida Romli (PhD )
SOC, UUM

**CORRECTNESS TESTING– Test Case Coverage for Structural Test Data Generation**

**Instruction:** Fill in each row with the result collected from the testing.

**A1: Question 1**

| Test Case (TC) | Input | Output | Test Set Description |
|---|---|---|---|
| 1 | | | |
| 2 | | | |
| 3 | | | |
| 4 | | | |
| 5 | | | |
| 6 | | | |
| 7 | | | |
| 8 | | | |
| 9 | | | |
| 10 | | | |
| 11 | | | |
| 13 | | | |
| 14 | | | |
| 15 | | | |
| 16 | | | |
| 17 | | | |
| 19 | | | |
| 20 | | | |

**A2: Question 2**

| Test Case (TC) | Input | Output | Test Set Description |
|---|---|---|---|
| 1 | | | |
| 2 | | | |
| 3 | | | |
| 4 | | | |
| 5 | | | |
| 6 | | | |
| 7 | | | |
| 8 | | | |
| 9 | | | |
| 10 | | | |
| 11 | | | |
| 13 | | | |
| 14 | | | |
| 15 | | | |
| 16 | | | |
| 17 | | | |
| 19 | | | |
| 20 | | | |

**A3: Question 3**

| Test Case (TC) | Input | Output | Test Set Description |
|---|---|---|---|
| 1 | | | |
| 2 | | | |
| 3 | | | |
| 4 | | | |
| 5 | | | |
| 6 | | | |
| 7 | | | |
| 8 | | | |
| 9 | | | |
| 10 | | | |
| 11 | | | |
| 13 | | | |
| 14 | | | |
| 15 | | | |
| 16 | | | |
| 17 | | | |
| 19 | | | |
| 20 | | | |
| 21 | | | |
| 22 | | | |
| 23 | | | |
| 24 | | | |
| 25 | | | |

**A4: Question 4**

| Test Case (TC) | Input | Output | Test Set Description |
|---|---|---|---|
| 1 | | | |
| 2 | | | |
| 3 | | | |
| 4 | | | |
| 5 | | | |
| 6 | | | |
| 7 | | | |
| 8 | | | |
| 9 | | | |
| 10 | | | |
| 11 | | | |
| 13 | | | |
| 14 | | | |
| 15 | | | |
| 16 | | | |
| 17 | | | |
| 19 | | | |
| 20 | | | |
| 21 | | | |
| 22 | | | |
| 23 | | | |
| 24 | | | |
| 25 | | | |
| 26 | | | |
| 27 | | | |
| 28 | | | |
| 29 | | | |
| 30 | | | |
| 31 | | | |
| 33 | | | |
| 34 | | | |
| 35 | | | |

Pusat Pengajian Pengkomputeran
School of Computing
Universiti Utara Malaysia

***Post-test* Experiment– Test Data Adequacy of Dynamic Structural Test Data Generation for Programming Assessment**

Dear Evaluators,

This *post-test* experiment intends to measure the degree of DyStruc TDG used in preparing a set of test data to perform the dynamic structural testing in assessing students programming exercises. All responses from this post-test experiment are anonymous and will be strictly confidential.

Thank you for your kind co-operation.

Kind Regards,
Md. Shahadath Sarker (816283)
MSc. IT (Course Work and Dissertation)

Supervisor:
Rohaida Romli (PhD )
SOC, UUM

**CORRECTNESS TESTING– Test Case Coverage for Structural Test Data Generation**

**Instruction:** Fill in each row with the result collected from the testing.

**A1: Question 1**

| Test Case (TC) | Input | Output | Test Set Description |
|---|---|---|---|
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |

**A2: Question 2**

| Test Case (TC) | Input | Output | Test Set Description |
|---|---|---|---|
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |

**A3: Question 3**

| Test Case (TC) | Input | Output | Test Set Description |
|---|---|---|---|
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |

**A4: Question 4**

| Test Case (TC) | Input | Output | Test Set Description |
|---|---|---|---|
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |