International Journal on Advanced Science Engineering Information Technology

Vol.10 (2020) No. 1
ISSN: 2088-5334

Automated Test Cases and Test Data Generation for Dynamic Structural Testing in Automatic Programming Assessment Using MC/DC

Rohaida Romli#1, Shahadath Sarker#, Mazni Omar#, Musyrifah Mahmod#

# *School of Computing, College of Arts and Sciences, Universiti Utara Malaysia, Kedah, Malaysia*
E-mail: 1aida@uum.edu.my

***Abstract*— Automatic Programming Assessment (or APA) is known as a method to assist educators in executing automated assessment and grading on students' programming exercises and assignments. Having to execute dynamic testing in APA, providing an adequate set of test data via a systematic process of test data generation is necessarily essential. Though researches respecting to software testing have proposed various significant methods to realize automated test data generation, it occurs that recent studies of APA rarely utilized these methods. Merely some of the limited studies appeared to resolve this circumstance, yet the focus on realizing test set and test data covering more thorough dynamic-structural testing are still deficient. Thus, we propose a method that utilizes MC/DC coverage criteria to support more thorough automated test data generation for dynamic-structural testing in APA (or is called DyStruc-TDG). In this paper, we reveal the means of deriving and generating test cases and test data for the DyStruc-TDG method and its verification concerning the reliability criteria (or called positive testing) of test data adequacy in programming assessments. This method offers a significant impact on assisting educators dealing with introductory programming courses to derive and generate test cases and test data via APA regardless of having knowledge of designing test cases mainly to execute structural testing. As regards to this, it can effectively reduce the educators' workload as the process of manual assessments is typically prone to errors and promoting inconsistency in marking and grading.**

***Keywords*— automatic programming assessment; test data generation; dynamic testing; structural coverage; MC/DC.**

I. Introduction

Generally, courses related to programming are aimed at developing student's knowledge and skill in solving programming problems and understanding related concepts and principles. These courses are more practical when students are given more programming exercises via laboratory hands-on and assignments. Since the number of students in a programming class are commonly large, it is very tough to assess the programming exercises and assignments manually by educators [1]. Manually assessing these programming exercises most commonly requires much effort and dealing with time-consuming tasks [2]. It also directly promotes unintentional biases and different standards of marking schemes [1] due to human deficiency. Thus, an effort to realize automated means of assessing and grading students' programming exercises or is formally known as Automatic Programming Assessment (or APA) contributes a significant role to the related educators as well as benefits students with instant feedback on their programming solutions [3]. Concerning this matter, quite a few tools that are so-called Automatic Programming Assessment Systems or APAS were developed to accommodate students and educators with APA. Among them include Assyst [4], BOSS [5], TRAKLA2[6], PASS [7], FEAT [8], a tool developed by University of Southampton [9] and others. Unfortunately, most of these APAS lacking a systematic means of generating test sets or test data automatically.

By default, APA necessitates test data to execute dynamic testing on students' program [1]. In software testing, manually or unsystematically generating test data is laborious, prone to errors, demanding, an expensive, and feasible task in practice [10]. Dealing with these concerns, some methods to realize automated test data generations have been proposed in the industry [11]-[16]. Activities dealing with the process of generating test data automatically are commonly referred to as Automated Test Data Generation or ATDG. In APA particularly, it happens to serve the same problematic issues. Even the programming assessments could be realized automatically in the most efficient way, the means of preparing test data most of the time remains as manual effort or depending on pre-set test data in files. However, it appears in the recent studies of

APA very few of them seemed to utilize these methods. Among of those studies that focus on structural testing include Ihantola [17], Tillman *et al*. [18], and Romli [19]. Path coverage criterion is identified as the most popular structural code coverage used to realize the process of test data generation for structural testing in APA.

In designing test cases or test data, particularly for structural testing, it is notable for ensuring that testing could achieve a certain level of thoroughness. This thoroughness will determine how adequate the testing is [20]. As adequate testing is typically counted on test adequacy criteria [21], hence it can distinguish between good and bad test cases and determine whether the testing is enough [22]. The means of designing test cases for structural testing is dealing with structural coverage criteria such as statement coverage, branch coverage, path coverage, condition/decision coverage, multiple condition coverage and Modified Condition/Decision Coverage (MC/DC) coverage [23].

Existing studies in APA that are focusing on the test data generation to derive and generate test data for structural testing (as previously mentioned) have not sufficiently included the ideal test criterion to derive adequate test data with a certain level of thoroughness. Regarding these issues, this study proposes an automated method of test data generation to execute dynamic-structural testing that integrates path testing coverage and Modified Condition/ Decision Coverage (MC/DC).

This paper is organized as follows: In section 2, the related works concerning integration of APA and ATDG are addressed. Section 3 layouts the detailed design of the DyStruct-TDG method. Section 4 provides a discussion on the evaluation and results obtained from the conducted controlled experiment to verify the adequacy of derived test data. Finally, Section 5 is a conclusion to this paper.

A. Related Work

Since the 1960s, APA has gained more attention from researchers in the area of Computer Science education and has continued to raise enormous attention until to date. Its crucial aim mainly to introduce and promote APAS that helps educators to lessen their workloads, to ensure the consistency of marking assessment items and taking account of thorough testing on students' programming solutions [19]. Moreover, APA practically allows immediate feedback without doing fewer exercises [17].

Based on a review done by Douce *et al.* [24], APAS are commonly grouped into three generations, which can be described from early assessment tools to Command-Line Interface (CLI) or Graphical User Interface (GUI) distributed systems and later as web-based systems. On top of that, Ihantola *et al.* [25] also provided a more detailed review, of which identified that most of the assessment tools were developed for one specific class or assignment which eventually forced the programming instructors or lecturers to develop their own the test data frameworks to observe the functionality and behavior of the programs they required for. Besides, a study conducted by Liang *et al.* [26] outlined that there were a small number of APAS that furnished rich, critical and timely response. Also, more details on the advantages of APAS have been outlined in a study conducted by Rahman and Nordin [27].

Even there have been some automated methods for test data generation available to support activities in software testing, yet existing studies of APA seldom adopt these methods systematically even the focus is structural testing. As previously mentioned in the prior section, it happens merely very limited studies have put attentions on integrating APA and ATDG where the emphasis is on structural testing. The following paragraphs provide briefly explanations on those related studies.

A study proposed by Ihantola [17] applied a software model checker named Java PathFinder (JPF) with utilizing a symbolic execution technique to generate test data to support structural testing. Also, Tillmann *et al*. [18] employed the similar technique named dynamic symbolic execution to generate the required test data.

Among the most recent study is as proposed by Romli [19]. The study proposes a framework of test data generation to execute APA which covered both testing categories: structural and functional testing. In realizing ideal test criterion, the framework embeds both the positive testing and negative testing criteria. In terms of the structural testing coverage, it is regarding path testing coverage criteria. Table 1 summarizes the described studies.

TABLE I
INTEGRATION OF APA AND ATDG (EXTENDED FROM ROMLI [19])

Author (Year)	**Type of Testing (Dynamic)**		**Test Data Generation Method**
	Functional	**Structural**	
Ihantola [17]	No	Yes	Symbolic execution with JPF
Tillman *et al.* [18]	No	Yes	Dynamic symbolic execution with Pex
Romli [19]	Yes	Yes	Positive and negative testing with path coverage and an integration of specification derived test and simplified boundary analysis techniques

Based on the prior explanation, it can be concluded that to date, only very few studies have made systematic efforts to integrate both APA and ATDG particularly to achieve structural testing. It is shown that the studies merely utilized the techniques to cover test data adequacy criteria which did not consider the thoroughness criteria of the generated test cases. By addressing this gap, this study proposes a method called DyStruc-TDG which adapting MC/DC criteria to derive and generate adequate test cases and test data to cover more thorough testing coverage.

II. MATERIAL AND METHOD

The DyStruc-TDG method was constructed by translating the selected structural coverage criteria (that is MC/DC criteria) into a design of test cases. The method only covers the positive testing criterion [28][29]. The following Fig. 1 illustrates an activity diagram that shows processes involved in the DyStruc-TDG method.

MC/DC is a structural coverage criterion that explains every condition separately change the result [20]. The decision gets hold of all potential outcomes at least once and covers both the true and false values. Each condition in the decision independently affects the decision's outcome. MC/DC is usually practical for vital systems which developed in the avionics field and one of the more practical criteria to be applied [23]. If it is given a segment of codes in Fig. 2, the required test cases to be executed to cover the MC/DC coverage is shown in Table 2.

The important aspect of MC/DC is testing should show the independent effect of atomic Boolean conditions on the Boolean expressions in which they occur [30]. Based on the mentioned MC/DC criteria, Table 3 concludes the formula for generating test cases by applying MC/DC coverage criterion by considering the minimum number of test cases that would apply to cover adequate test cases particularly for APA. However, in the worst-case scenario, the formula obtained will be $2 \times N$ (example as shown in Table 2).

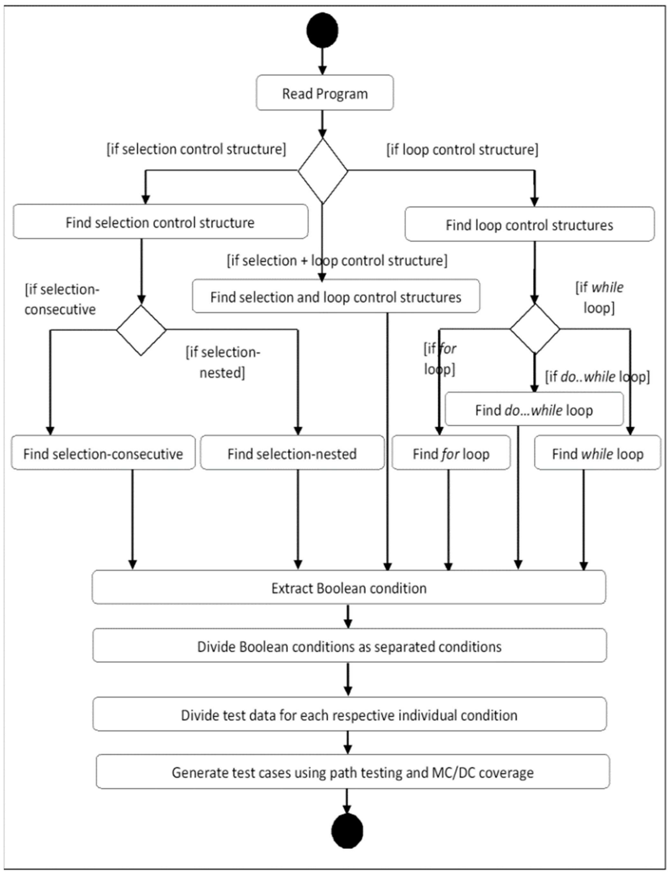

Read Program
[if selection control structure]
[if loop control structure]
Find selection control structure
Find loop control structures
[if selection + loop control structure]
Find selection and loop control structures
[if selection-consecutive]
[if while loop]
[if selection-nested]
[if for loop]
[if do..while loop]
Find do...while loop
Find selection-consecutive
Find selection-nested
Find for loop
Find while loop
Extract Boolean condition
Divide Boolean conditions as separated conditions
Divide test data for each respective individual condition
Generate test cases using path testing and MC/DC coverage

Fig. 1 An activity diagram showing processes involved in dystruc-TDG method

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

Fig. 2 Code segment for selection control structures (MC/DC)

In order to demonstrate how the design of the test set in the DyStruc-TDG method is mapping to APA, the sub-sequence paragraphs provide the related details. Fig. 3 depicts the sample of programming exercise; Table 4 shows its respective Boolean expressions with their True and False values and Table 5 summarizes the derived test cases and test data. Based on Table 3, as the number of options is two (see Table 4), hence 4 test cases will be generated based on the rule of truth table (see Table 6). As MC/DC only considers test cases that independently effect on each option (see Table 7 and Table 8), hence 3 test cases as shown in Table 5 are selected. By referring Table 6, TC4 is excluded.

TABLE II
TEST CASES GENERATION USING MC/DC

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)					

TABLE III
THE FORMULA OF DERIVING TEST CASES OBTAINED FROM MC/DC

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

Fig. 4 illustrates an example of programming exercise with *do…while* loop. Table 9 shows its respective Boolean expressions with their True and False values of each respective variable involved and Table 10 summarizes the derived test set and test data. This study adapts loop coverage criteria to guide in deriving the required test cases. Loop coverage (is also called loop boundary adequacy) ensures that for every loop, there are test cases to test the loop so that it iterates zero time, at least once, and more than once [31]. As APA does not require pervasive testing, hence in the DyStruc-TDG method, we only consider test cases to cover zero and more than one iteration. Thus, only two test cases are considered in this case, as it merely involves one option (see Table 10).

The DyStruc-TDG method can generate test data for various data types such as integer, real numbers, character, String, Boolean and others, based on the True and False values of an individual Boolean expression. The True value is derived based on the value at the boundary of a given Boolean expression. It is realized by directly extracting the relational and logical operator and value defined as part of the Boolean expression (see Table 4 and Table 9). For example, if the Boolean expression is *(age>=20)*, the True value is 20. On the other hand, the False value is generated by finding among the nearest value as extracting from the given Boolean expression (that is outside from the given boundary). For example, if the Boolean expression is *(age>=20)*, the false value is 18.

```
import java.util.Scanner;
public class test{
  public static void main (String[]args){
       int age; char city;
       Scanner scan =new Scanner (System.in);
       System.out.println(" Enter your age: ");
       age= scan.nextInt();
       System.out.println(" Enter city: ");
       city= scan.next().charAt(0);
       if ((age>=20)&&(city=='K')){
         System.out.println(" You can vote" );
       }
       else{
         System.out.println(" You are not
           qualified" );
       }
  }
}
```

Fig. 3 Sample of Programming Exercise with Two Variables as Part of Its Boolean Expressions.

TABLE IV
BOOLEAN EXPRESSION WITH THE TRUE AND FALSE OF EACH RESPECTIVE VARIABLES INVOLVED

Expression/Option	**Variable**	**True value**	**False value**
(age>=20) = A	age	20	18
(city=='K') = B	city	'K'	'A'

TABLE V
GENERATED TEST CASES AND TEST DATA FOR THE TWO OPTIONS AS SHOWN IN TABLE IV

Test Case	**Option (A)**	***age***	**Option (B)**	***city***
TC 1	True	20	True	'K'
TC 2	False	18	True	'K'
TC 3	True	20	False	'A'

TABLE VI
GENERATED TEST CASES BASED ON TRUTH TABLE

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

TABLE VII
TEST CASES WITH INDEPENDENTLY EFFECT ON OPTION (A)

Option (A)	**Option (B)**	**Option (A&&B)**	**Test Case (from Table 6)**
True	True	True	TC 1
False	True	False	TC 3

TABLE VIII
TEST CASES WITH INDEPENDENTLY EFFECT ON OPTION (A)

Option (A)	**Option (B)**	**Option (A&&B)**	**Test Case (from Table 6)**
True	True	True	TC 1
True	False	False	TC 2

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

Fig. 4 Sample of Programming Exercise for *Do…While* Loop with One Variable as Part of Its Boolean Expression.

TABLE IX
BOOLEAN EXPRESSION WITH THE TRUE AND FALSE OF EACH RESPECTIVE VARIABLES INVOLVED

Expression/Option	**Variable**	**True Value**	**False Value**
(x<50) = A	*x*	48	52

TABLE X
GENERATED TEST CASES AND TEST DATA FOR THE TWO OPTIONS AS SHOWN IN TABLE IX

Test Case	**Option (A)**	***x***
TC1	**True**	48
TC 2	**False**	52

III. RESULTS AND DISCUSSION

A controlled experiment that utilizes the one-group pretest and post-test design [32] were conducted in this study. This experiment aimed to verify the completeness coverage of test data adequacy-reliability on the DyStruc-TDG method. The subjects of the controlled experiment were among the lecturers who have been teaching the programming courses in one of the public universities in the northern region of Malaysia. Due to the different teaching schedules of the lecturers, the experiment was conducted as multi-shot sessions individually. The data were collected from the subjects only at one time instead of collecting several times. In this experiment, the number of subjects was ten (10).

The controlled experiment used a set of *pre-test* and *post-test* questions to collect the required data. The set of *pre-test* and *post-test* questions have consisted of the same contents. Four samples of programming exercises were used as the assignments of which cover the main two control structures (selection and repetition). One question to cover the selection, two questions with regard repetition (counter-loop and sentinel-loop) and the remaining one is an integration of the selection and repetition control structures. Each exercise was provided with its solution model.

This study also conducted a comparative evaluation to provide a comparative analysis of the coverage of test cases and test data between the DyStruc-TDG method and other methods proposed by previous researchers. There are three studies were selected for the comparison with the DyStruc-TDG method: Ihantola [17] and Tillmann *et al*. [18] and Romli [19]. A sample of programming exercise was used as a benchmark for the comparison. In this article, particularly for the controlled experiment, its result only will cover two of the provided programming exercises (selection and repetition). The following paragraphs include the analysis and results of the mentioned evaluations.

For the selection control structure, the programming exercise used in the controlled experiment, its control flow (in an activity diagram) can be illustrated as in Fig. 5. Table 8 shows the paths covered for the selection control structure which consisted of 3 paths (path 1, path 2 and path 3). Table 11 has listed the number of test cases to cover the 3 paths by each subject. The mean of deriving test cases and test data were based on as has been applied in current practice (or Current Method). Table 12 shows the total of test cases produced by each subject considering all the 3 paths.

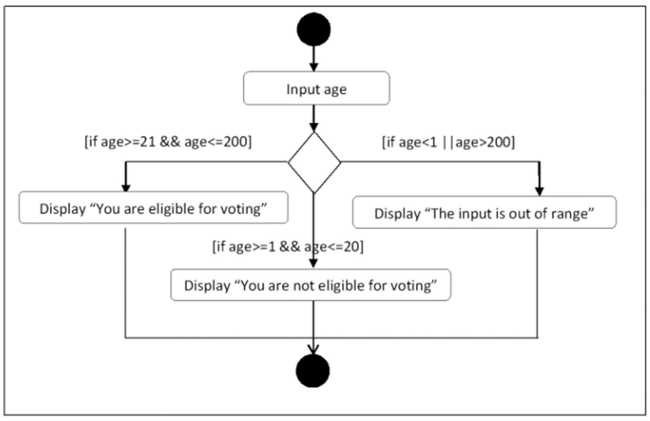

Input age
[if age>=21 && age<=200]
[if age<1 ||age>200]
Display "You are eligible for voting"
Display "The input is out of range"
[if age>=1 && age<=20]
Display "You are not eligible for voting"

Fig. 5 An Activity Diagram Showing the Control Flow of a Selection Control Structure.

TABLE XI
NUMBER OF PATHS FOR THE SELECTED SELECTION CONTROL STRUCTURE

Path 1	**Path 2**	**Path 3**
if *(age>=21 && age <= 200)* is True	if *(age>=1 && age <= 20)* is True	if *(age<1 && age > 200)* is True

TABLE XII
NUMBER OF TEST CASES BY EACH SUBJECT BASED ON EACH PATH

Subjects	**Path 1 (Test Cases)**	**Path 2 (Test Cases)**	**Path3 (Test Cases)**	**Total of Test Cases**
1	2	4	0	6
2	3	2	2	7
3	1	2	0	3
4	3	4	3	10
5	2	2	2	6
6	3	3	2	8
7	4	2	4	10
8	4	4	2	10
9	1	1	4	6
10	1	0	4	5

In the case of the DyStruc-TDG method, it generates test cases in a consistent way for each path by considering each Boolean expression individually based on MC/DC coverage concept. For example, Path 1 (*if (age>=21&& age <=200)* is true) has two Boolean expressions (*age>=21*) and (*age <=200*). Based on MC/DC formula, the DyStruc-TDG method generates 3 test cases. Also, both of Path 2 and Path 3 also have 2 Boolean expressions. Concerning this, the DyStruc-TDG method also generates 3 test cases for each path. Table 13 has listed the individual test cases derived by the DyStruc-TDG method and the total of test cases produced by the method considering all the 3 paths.

A line graph from Fig. 6 shows a comparison between the number of test cases derived by the subjects and the DyStruc-TDG method. The DyStruc-TDG method consistently generated test cases and covered all the paths. Based on MC/DC criteria, the DyStruc-TDG method considers each Boolean expression individually and at the same time, has fully covered path testing coverage. On the other hand, the subjects appeared to derive test cases inconsistently and indeed, some subjects did not cover all the paths (see Table 9). Also, they did not consider each Boolean expression individually as what MC/DC criteria did.

TABLE XIII
NUMBER OF TEST CASES DERIVED BY DYSTRUCT-TDG METHOD BASED ON EACH PATH

Subjects	**Path 1 (Test Cases)**	**Path 2 (Test Cases)**	**Path3 (Test Cases)**	**Total of Test Cases**
1	3	3	3	9
2	3	3	3	9
3	3	3	3	9
4	3	3	3	9
5	3	3	3	9
6	3	3	3	9
7	3	3	3	9
8	3	3	3	9
9	3	3	3	9
10	3	3	3	9

From the line graph, the subjects (4, 7, and 8) derived one extra test case then the DyStruc-TDG method. In this case, the DyStruc-TDG method reduces one test case. On the other hand, other subjects appeared to derive fewer test cases than the DyStruc-TDG method. Although the related subjects produced fewer test cases, they seemed likely did not cover all the paths and each Boolean expression individually. Thus, this result concludes that the subjects happened to did not deriving consistent test cases to provide thoroughness testing as compared to the DyStruc-TDG method of which covers all the paths and Boolean expressions individually and provides the thoroughness of testing. The results of other programming exercises happened to have similar patterns as for the selection control structure. However, for the sentinel-loop control structure, DyStruc-TDG method came out with a smaller number of test cases that are considered adequate and does cover thoroughness testing. The sub-sequence paragraphs provide a brief explanation of the means of deriving test cases and test data for the sentinel-loop control structure.

For the sentinel-loop control structure, the programming exercise used in the controlled experiment, its control flow (in an activity diagram) can be illustrated as in Fig. 7. The question was about the repetition control structure for a sentinel loop which consisted of one path with a Boolean expression (*number! = 0*). Table 14 shows the number of derived test cases as has been applied in current practice (Current method). It is shown that various patterns in the means of deriving test cases across different subjects.

In the case of the DyStruc-TDG method, like the selection control structure, it consistently generates test cases for the identified path by considering the included Boolean expression (*number! =0).* Based on MC/DC formula (as mentioned in the prior section), the DyStruc-TDG method generates 2 test cases to cover both True and False values as only test cases to cover zero and more than one iteration have been considered. As compared to the test cases derived based on the Current Method, it happened some subjects derived more test cases for False values, whereby only one value is enough to cover adequate testing. Some others happened to consider more test cases covering for True values. These inconsistent situations possibly will increase the number of test cases derived when the number of options increases.

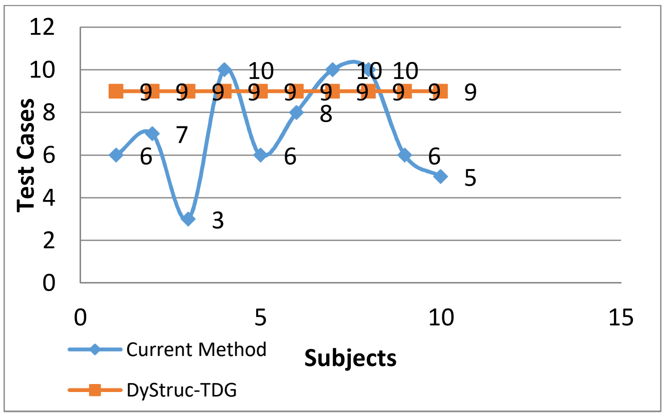


Fig. 6 Test Cases Coverage between the Current Practice (as Employed by the Subjects) and DyStruc-TDG Method for a Selection Control Structure.

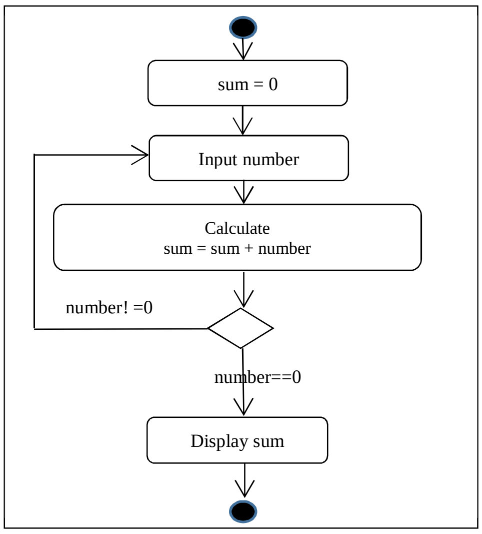


Fig. 7 An Activity Diagram Showing the Control Flow of a Repetition (sentinel-loop) Control Structure.

TABLE XIV
NUMBER OF TEST CASES BY EACH SUBJECT BASED ON PATH *NUMBER!=0*

Subjects	Path *number! =0* (Test Cases)
1	4
2	3
3	4
4	6
5	3
6	5
7	4
8	6
9	3
10	3

By referring to Table 14, Subject (4) derived a total of 6 test cases, where 1 test case is for a True value and another 5 test cases are for False values. A similar pattern has been shown by Subject (6) in which he/she derived 4 redundant test cases merely to cover False value. Based on the loop boundary adequacy, one test case is adequate to cover False value regardless of any data type. A line graph from figure 8 shows a comparison between the number of test cases derived by the subjects and the DyStruc-TDG method for the programming exercise shown in Fig. 9. Again, it can be concluded that the DyStruc-TDG method is able to derive and generate test cases and test data in consistent way and covered all the necessary paths.

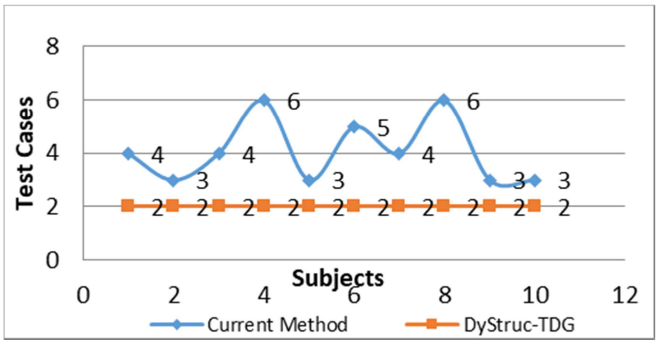
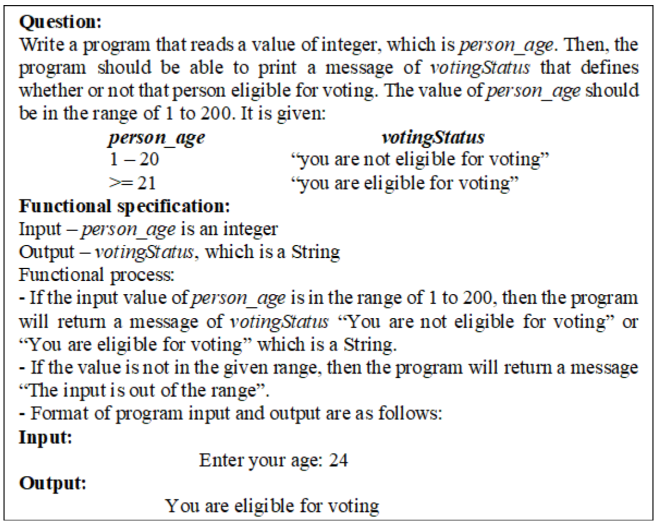


Fig. 8 Test Cases Coverage between the Current Practice (as Employed by the Subjects) and DyStruc-TDG Method for a Repetition Control Structure.

As previously mentioned, this study also conducted a comparative evaluation to compare in terms of test data adequacy for structural testing. The comparison was done among three studies in structural testing namely Ihantola [17] and Tillmann *et al*. [18]; Romli[19] and DyStruc-TDG. Fig. 9 depicts a sample of programming exercises used for this comparison.



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

Fig. 9 Sample of Programming Exercise Used for Comparative Evaluation

The following Table 15 shows the result of a comparison between the three selected studies of which the focus is on structural testing. Based on the comparative evaluation, it shows that Ihantola[17] and Tillmann *et al*.[18] and Romli[19] have derived respectively 4 and 5 test cases. On the other hand, the DyStruc-TDG method has derived 9 test cases. However, in terms of the structural testing coverage, as DyStruc-TDG method utilized MC/DC criteria, hence it covers each Boolean expression individually. The study proposed by Romli [19] requires human involvement to assign test data for the generated test cases whereby the DyStruc-TDG method able to derive and generate test cases and test data automatically. Thus, it concludes that the DyStruc-TDG method has covered more thoroughness testing than another three studies proposed by Ihantola [17] and Tillmann *et al*. [18] and Romli[19].

TABLE XV
RESULT OF THE COMPARATIVE EVALUATION

Criteria of comparison	**Ihantola [17] and Tillmann et al. [18]**	**Romli [19]**	**DyStruc-TDG**
Number of test cases	4	5	9
Test data coverage	1. *age >= 21 && age <= 200* 2. *age >= 1 && age <= 20* 3. *age < 1 && age > 200*	1. *age >= 21 && age <= 200* 2. *age >= 1 && age <= 20* 3. *age < 1* 4. *age > 200* 5. Illegal path condition	1. *age >= 21 && age <= 200* 2. *age >= 1 && age <= 20* 3. *age < 1* 4. *age > 200*
Values of test data	Input parameter based on path condition	Lecturer need to assign test data	Automated generated

IV. CONCLUSION

This paper has presented the means of deriving adequate test cases and test data of which covering the thoroughness testing. The method is called DyStruc-TDG method to cover dynamic structural testing in APA. As to realize the method by way of a tangible deliverable a test data generator prototype was developed. An evaluation to verify the completeness coverage of the criteria test data adequacy-reliability of the proposed method was conducted via the developed test data generator. The verification was done by comparing the derived test cases and test data produced by the educators who have been teaching introductory programming courses as compared to those produced by the DyStruc-TDG method. The results obtained from this verification revealed that the DyStruc-TDG method can derive the desired test cases and test data that do satisfy the test data adequacy criteria and enough thoroughness testing level. Overall, this method is significantly able to assist educators and instructors of introductory programming courses to derive and generate an adequate set of test data automatically regardless of having any detailed knowledge in designing test cases for structural testing.

ACKNOWLEDGMENT

We would like to acknowledge the Ministry of Higher Education Malaysia - FRGS Fund (S/O Code: 12821) of Universiti Utara Malaysia for supporting this work.

REFERENCES

[1] R. Romli, S. Sulaiman, K. Z. Zamli, “Automatic programming assessment and test data generation a review on its approaches”. In Proceedings of Information Technology (ITSim) International Symposium 3, 2010, pp.1186-1192.

[2] D. Jackson, “A Software System for Grading Student Computer Programs”, *Computers and Education*, 27 (3-4), pp. 171-180, 1996.

[3] R. Saikkonen., L. Malmi, A. Korhonen, “Fully Automatic Assessment of Programming Exercises”, ACM SIGCSE Bulletin, 33 (3), 2001, pp.133-136. R. E. Sorace, V. S. Reinhardt, and S. A. Vaughn, “High-speed digital-to-RF converter,” U.S. Patent 5 668 842, Sept. 16, 1997.

[4] D. Jackson, M. Ushe, “Grading student programs using ASSYST”, Proceedings of the 28th SIGCSE Technical Symposium on Computer Science Education, San Jose, CA., 1997, pp. 335–339.

[5] M. Luck, M. S. Joy, “A secure on-line submission system”, *Journal of Software – Practise and Experience*, 29 (8), pp. 721-740, 1999.

[6] L. Malmi, V. Karavirta,, A. Korhonen, J. Nikander, O. Seppala, P. Silvasti, “Visual Algorithm Simulation Exercise System with Automatic Assessment: TRAKLA2”, *Informatics in Education*, 3(2), pp. 267-288, 2004.

[7] M. Choy, U. Nazir, C.K Poon, Y.Y Yu, “Experiences in Using an Automated System for Improving Students’ of Computer Programming” , Advances in Web-Based Learning – ICWL 2005, Lecture Notes in Computer Science, Vol. 3583/2005, 2005, pp. 267–272.

[8] T. Tang, R. Smith, J. Warren, S. Rixner, “Data-Driven Test Case Generation for Automated Programming Assessment”, Proceedings of the 2016 ACM Conference on Innovation and Technology in Computer Science Education ITiCSE 16, 2016, pp. 260-265.

[9] H. Fangohr, N. O'Brien, A. Prabhakar, A. Kashyap, “Teaching Phyton Programming with Automatic Assessment and Feedback Provision”, arXiv:1509.03556 [cs.CY], 2015, pp. 1-26.

[10] S. Monpratarnchai, S. Fujiwara, A. Katayama, T. Uehara, “Automated Testing for Java Programs using JPF-based Test Case Generation, ACM SIGSOFT Software Engineering Notes, 39 (1), 2014, pp. 1-5.

[11] L. A. Clarke, “A system to generate test data and symbolically execute programs”, *IEEE Transaction on Software Engineering*, SE-2(3), pp. 215-222, 1976.

[12] N. Gupta, A.P Mathur, M. L Soffa, “Automated Test Data Generation Using an Iterative Relaxation Method”, *ACM SIGSOFT Software Engineering Notes*, 23 (6), pp. 231-245, 1998.

[13] J. Offutt, S. Liu, A. Abdurazik, P. Ammann, “Generating Test Data from State-Based Specifications”, *Software Testing, Verification and Reliability*, Vol. 13, pp. 25–53, 2003.

[14] K.Z. Zamli,. A. M. Isa, M. F. J. Klaib, S.N. Azizan, “Tool for Automated Test Data Generation (and Execution) Based on Combinatorial Approach”, *International Journal of Software Engineering and Its Applications*, 1(1), pp. 19-36, 2007.

[15] W. Zidoune, T. Benouhiba, “Targeted adequacy criteria for search-based test data generation”, International Conference on Information Technology and E-Services, 2012, pp. 1-6.

[16] R.P. Pargas, M. J. Harrold, R. Peck, “Test-Data Generation Using Genetic Algorithms”, *Journal of Software Testing, Verification and Reliability*, 9(4), pp. 63-282, 1999.

[17] P. Ihantola, “Test Data Generation for Programming Exercises with Symbolic Execution ind Java PathFinder”, Proceedings of the 6th Baltic Sea Conference on Computing Education Research: Koli Calling 2006, 2006, pp. 87 – 94.

[18] N. Tillmann, J. D Halleux, T. Xie, S. Gulwani, J. Bishop, “Teaching and Learning Programming and Software Engineering via Interactive Gaming”, Proceedings of the 2013 International Conference on Software Engineering (ICSE’13), San Francisco,CA, USA, 2013, pp. 1117-1126.

[19] R. Romli, “Test Data Generation Framework for Automatic Programming Assessment”, PhD Thesis, Universiti Sains Malaysia, Malaysia, 2014.

[20] K. J. Hayhurst, D. S. Veerhusen, J.J. Chilenski,, L.K Rierson, “A practical tutorial on modified condition/decision coverage”, NASA STI Report Series, 2001.

[21] H. Zhu, P.A. V. Hall, J. H. R May, “Software Unit Test Coverage and Adequacy”, *ACM Computing Surveys*, 29 (4), pp. 365-427, 1997.

[22] H. Zhu, “Axiomatic Assessment of Control Flow-based Software Test Adequacy Criteria”, *Software Engineering Journal*, 10 (5), pp. 194 -204, 1995.

[23] K. Ghani, J.A Clark, “Automatic Test Data Generation for Multiple Condition and MCDC Coverage”, Proceedings of the 2009 Fourth International Conference on Software Engineering Advances, 2009, pp. 152 -157.

[24] C. Douce, D. Livingstone & J. Orwell, J. “Automatic test-based assessment of programming: A review”, *Journal on Educational Resources in Computing (JERIC), 5*(3), Aticle No. 4, 2005.

[25] P. Ihantola, T. Ahoniemi, V. Karavirta & O. Seppälä, O. “Review of recent systems for automatic assessment of programming assignments”, In *Proceedings of the 10th Koli Calling International Conference on Computing Education Research*, 2010, pp. 86-93.

[26] PY. Liang, Q. Liu, J. Xu & D. Wang, D. “The recent development of automated programming assessment”. Proceedings of International Conference on Computational Intelligence and Software Engineering (CiSE 2009), 2009, pp. 1-5.

[27] K. A. Rahman & M. J. Nordin, A review on the static analysis approach in the automated programming assessment systems. Proceedings of National Conference on Programming, 2007, Vol. 7.

[28] IPL Information Processing Ltd. Designing Unit Test Cases, 1997. Available: http://www.ipl.com/pdf/p0829.pdf. Retrieved on: 10 Feb 2009.

[29] J. Watkins, S. Mills, *Testing IT: An Off-the-Shelf Software Testing Process*, 2nd Edition, 2011, Cambridge University Press, NY, USA.

[30] S. Rayadurgam, M.P.E. Heimdahl, “Generating MC/DC Adequate Test Sequences Through Model Checking”, Proceedings of the 28th Annual IEEE/NASA Software Engineering Workshop -- SEW-03. Greenbelt, Maryland, 2003, pp. 1–5.

[31] M. Pezze, M. Young, Software Testing and Analysis: Process, Principles, and Techniques, 2008, John Wiley & Sons, Inc, USA.

[32] J.R. Fraenkel, N.E Wallen, *How to Design and Evaluate Research in Education*, 4th Edition, 2000, McGraw-Hill Companies, Inc, U.S.A.