Jurnal
Teknologi

Full Paper

IMPROVING THE RELIABILITY AND VALIDITY OF TEST DATA ADEQUACY IN PROGRAMMING ASSESSMENTS

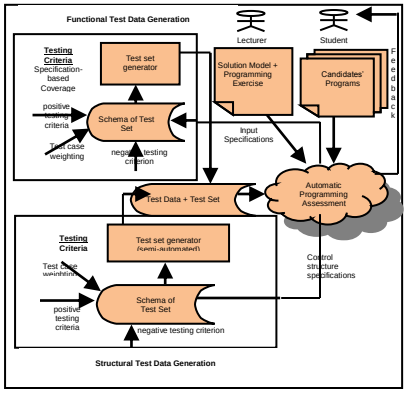
Rohaida Romli[a*], Shahida Sulaiman[b], Kamal Zuhairi Zamli[c]

[a]School of Computing(SOC), College of Arts and Sciences, Universiti Utara Malaysia, 06010 UUM Sintok, Kedah, Malaysia
[b]Faculty of Computing, Universiti Teknologi Malaysia, 81310 UTM Johor Bahru, Johor, Malaysia
[c]Faculty of Computer System and Engineering, Universiti Malaysia Pahang, Lebuhraya Tun Razak, 26300 Gambang, Kuantan, Pahang, Malaysia

Article history
Received
2 February 2015
Received in revised form
8 October 2015
Accepted
12 October 2015

*Corresponding author
aida@uum.edu.my

Graphical abstract



Abstract

Automatic Programming Assessment (or APA) has recently become a notable method in assisting educators of programming courses to automatically assess and grade students' programming exercises as its counterpart; the typical manual tasks are prone to errors and lead to inconsistency. Practically, this method also provides an alternative means of reducing the educators' workload effectively. By default, test data generation process plays an important role to perform a dynamic testing on students' programs. Dynamic testing involves the execution of a program against different inputs or test data and the comparison of the results with the expected output, which must conform to the program specifications. In the software testing field, there have been diverse automated methods for test data generation. Unfortunately, APA rarely adopts these methods. Limited studies have attempted to integrate APA and test data generation to include more useful features and to provide a precise and thorough quality program testing. Thus, we propose a framework of test data generation known as FaSt-Gen covering both the functional and structural testing of a program for APA. Functional testing is a testing that relies on specified functional requirements and focuses the output generated in response to the selected test data and execution, Meanwhile, structural testing looks at the specific program logic to verify how it works. Overall, FaSt-Gen contributes as a means to educators of programming courses to furnish an adequate set of test data to assess students' programming solutions regardless of having the optimal expertise in the particular knowledge of test cases design. FaSt-Gen integrates the positive and negative testing criteria or so-called reliable and valid test adequacy criteria to derive desired test data and test set schema. As for the functional testing, the integration of specification-derived test and simplified boundary value analysis techniques covering both the criteria. Path coverage criterion guides the test data selection for structural testing. The findings from the conducted controlled experiment and comparative study evaluation show that FaSt-Gen improves the reliability and validity of test data adequacy in programming assessments.

Keywords: Automatic Programming Assessment (APA), test data generation, functional testing, structural testing, test data adequacy, positive testing, negative testing

Abstrak

Kebelakangan ini, Penaksiran Pengaturcaraan Automatik (atau APA) umumnya dikenali sebagai suatu kaedah dalam membantu para pengajar kursus pengaturcaraan untuk melaksanakan penaksiran dan penggredan latihan-latihan pengaturcaraan pelajar secara automatik sebagai alternatif kepada: tugas-tugas penaksiran secara manual yang secara tipikalnya cenderung kepada penghasilan ralat dan ketidakseragaman penaksiran. Secara praktikal, kaedah ini juga adalah suatu alternatif yang efektif kepada

pengurangan beban tugas para pengajar. Secara asasnya, proses penjanaan data ujian memainkan peranan yang penting untuk melaksanakan pengujian dinamik terhadap aturcara pelajar. Pengujian dinamik melibatkan pelaksanaan suatu aturcara terhadap kepelbagaian input atau data ujian dan perbandingan antara perolehan dan output jangkaan perlu memenuhi spesifikasi aturcara. Terdapat pelbagai kaedah untuk mengautomasikan data ujian khususnya dalam bidang pengujian perisian. Walaubagaimanapun, kajian APA sedia ada jarang mengapliksikan kaedah ini. Didapati terdapat beberapa kajian yang amat terhad telah cuba mengintegrasikan APA dan penjanaan data ujian untuk menyediakan ciri-ciri kebergunaan yang lebih baik dan kualiti pengujian aturcara yang lebih jitu dan terperinci. Oleh itu, kami mencadangkan rangka kerja penjanaan data ujian yang dikenali sebagai *FaSt-Gen* untuk merangkumi kedua-dua pengujian aturcara fungsian dan berstruktur untuk APA. Pengujian fungsian adalah suatu pengujian yang bergantung kepada keperluan fungsian yang telah dikenalpasti dan memfokus kepada output yang dijana terhadap data ujian dan perlaksanaan yang telah dipilih. Manakala, pengujian berstruktur melihat kepada logik aturcara yang spesifik untuk menentusahkan bagaimana aturcara tersebut berfungsi. Secara keseluruhannya, *FaSt-Gen* menyumbang suatu kaedah kepada para pengajar kursus pengaturcaraan untuk menyediakan set data ujian yang berkecukupan untuk melaksanakan APA tanpa memerlukan kepakaran yang optimal dalam pengetahuan reka bentuk kes-kes ujian. *FaSt-Gen* mengintegrasikan kriteria pengujian positif dan negatif (atau kriteria kecukupan ujian boleh-dipercayai dan sahih) untuk menjana data ujian dan skema set ujian. Untuk pengujian fungsian, integrasi teknik *specification-derived test* dan *simplified boundary value analysis* merangkumi kedua-dua kriteria tersebut. Manakala, *path coverage criterion* memacu pemilihan data ujian untuk pengujian struktur. Dapatan daripada eksperimen yang telah dilaksanakan di kalangan pengajar menunjukkan bahawa *FaSt-Gen* berjaya memperbaiki kriteria kebolehpercayaan dan kesahihan kecukupan data ujian dalam penaksiran pengaturcaraan..

Kata kunci: Penaksiran pengaturcaraan automatik (APA), penjanaan data ujian, pengujian fungsian, pengujian struktur, kecukupan data ujian, pengujian positif, pengujian negatif

1.0 INTRODUCTION

Computer programming is prominently known as a complex intellectual activity and the core skill for the first year students pursuing a degree in Information Technology (IT) [1] as well as to other related disciplines such as Computer Science, Software Engineering and Engineering. Thus, any computer courses that have been offered in particular are deeply practical courses with the goal to develop students' understanding of the programming principles. To achieve that, a lot of programming exercises are given to students as hands-on or take-home assignments to ensure students' are consistent with the effectiveness on principles and concepts of programming in their learning process. In huge class sizes in Computer Science or IT programme, where there might be hundreds of students in a single course [2], the practice of programming exercises assessment leads to extensive workload to lecturers or instructors particularly if it has to be carried out manually. Apparently, manual assessment such as marking printed solutions by hand, which is time-consuming and requires much effort and attention is prone to error (s) at any levels of assessment [3]. It also may allow unintended biases and different standard of marking schemes. Furthermore, the feedback provided to students through marking is generally imited, and often late and outdated, particularly with the topic dealt in the assignment [4]. Thus, Automatic Programming Assessment (APA) has become an important method for grading students' exercises and giving feedback to them [5]. Practically, APA offers important benefits in terms of immediate feedback, objectivity and consistency of the evaluation as well as a substantial time saving in the evaluation of the assignments [6] without the need to reduce exercises [7]. Besides, it improves the consistency, accuracy and efficiency of the assessment [8].

To date, a number of automatic tools to cover both static and dynamic assessments for APA, which are called Automated Programming Assessment Systems (APAS) have been developed and tested for decades. The existing APAs include Assyst [8], BOSS [9], GAME [10], TRAKLA2 [11], PASS [12], ELP [1], CourseMaster [13], WeBWorK-JAG [14], SAC [15], Oto [16], ICAS [17], PETCHA [18], eGrader [19], and Bottlenose [20]. These systems provide advantages not only to lecturers, but might also play an important role in students' learning outcomes [21].

Typically, dynamic correctness assessment of students' programs involves the process of program quality testing through the execution of program with a range of test data and monitoring it conformance

through the comparison between the outputs produced and the expected ones [22]. Correctness is one of the quality attributes that can be defined as the degree to which the program performs it intended functions [23]. Since proving programs correctness is impractical in the practice of programming courses, running them with test data can allow some estimates of correctness to be made [24]. Generally, a program is considered correct if it consistently produces the right output [25]. Test data is defined as the inputs that have been devised to test the software [26]. The values must collectively satisfy some test data selection criteria or so-called test adequacy criteria [27]. According to Jackson [28], the process of applying the program to a number of sets of test data poses a few problems including the difficulty in devising a fullproof approach to judging whether the outputs produced by the program are correct; and possibly in some circumstances any two solutions will generate precisely the same output. Hence, the selection of a representative of test data should be carefully derived to perform accurate and efficient assessments. In addition, the criteria of selecting the test data should guarantee that it is able to conform to good test coverage of the solution model specification (or an assignment specifications) as well as to students' programs. Its main purpose is to avoid misleading feedback that can possibly cause misconception to the interpretation of the final assessment result.

In software testing research, various studies propose automated methods for test data generation [29-39]. Despite the potentials of the proposed methods in providing the most efficient way to generate test data for large-scale projects, researches in APA seldom adopt these methods. It appears that most of the studies in APA usefully execute tests and evaluate test results automatically, much of which have not sufficiently and systematically automate the generation of test. To date, very limited studies have attempted to incorporate both automation of test data generation and programming assessment (see Section 2). We intend to enhance the previous studies as the methods proposed are merely applied as a simple technique to generate test data, or the techniques require high technical skill to code particular run tests that limit its use for advanced users, or they derive test data based on only the functional or structural aspects of a program separately. Therefore, it motivates us to propose a framework of test data generation to derive and generate an adequate set of test data to perform the dynamic functional and structural testing of a program executed for APA or FaSt-Gen.

This paper consists of five consecutive sections after the Introduction. Section 2 reviews on studies that have attempted to incorporate both automation on test data generation and programming assessment. In the following section, it demonstrates how the test set allocates an adequate set of test data based on sample of programming exercises to map between FaSt-Gen and APA. Section 4 briefly lays out the overall design of FaSt-Gen in relation to APA. Section 5 reveals the analysis and findings from the conducted controlled experiment and comparative study to evaluate the completeness coverage of FaSt-Gen in term of reliability and validity test data adequacy. Finally, Section 6 concludes the paper.

2.0 RELATED WORK

To date, integration of automation of both test data generation and programming assessments has been at its initial stage. Some limited studies such as Guo *et al.* [40] and Cheng *et al.* [41] have attempted to automate the part of generating test data in APA. Each study seems to utilise an external tool that is either a product of past research within the same institution or a commercialized product that incurs cost. On the other hand, the studies by Jones [42] and Isong [43] employ certain systematic techniques of designing test cases. However, they still use manual way of deriving test data and generate the test data in a data file to be accessed by students. Besides, a number of studies [4][44][45][14][15][46][16][47] utilised JUnit framework to derive test set. The framework requires high technical skill that it requires rubric in mind, thus, it is not suitable for novices.

Thus far there have been five studies that mainly focus on integrating test data generation and APA that exclude the use of particular lecturers' knowledge in test cases design. The studies are among those that are relevant to the focus of this study. The study by Malmi *et al.* [11] introduced a tool known as *TRAKLA2* that is comprised of a framework to support randomized input values to assess students' programming assignments. However, randomized input is commonly arguable since it does not guarantee correctness of the program. Furthermore, the range of behaviours covered for large programs is often smaller in comparison to all possible behaviours of the program.

Shukur *et al.* [48] has proposed a schema to generate test data and test weight for marking students' program. It utilizes a boundary value analysis technique to generate test data. The proposed schema, however could not function well if there is a significant increase in the number of input variables. Such circumstances lead to a large combination of test data. Besides, this work merely applied functional based test data generation technique in deriving the desired test data.

Ihantola [49] worked on a study to propose a novel idea of extracting test schemas and test data. The technique of symbolic execution in Java PathFinder (JPF) software model checker was used in deriving the test data. The proposed technique not only produces a test set, but also a set of test patterns. Although the results of the study were reasonable and well applied in other contexts than automatic assessment of programming exercises, this work was just the first step to bring formally justified test data

generation and education closer to each other. In fact, the study only proposed a structural based test data generation technique.

A study by Tilmann *et al.* [50] focuses more on an interactive-gaming-based teaching and learning for introductory to advanced programming or software engineering courses. However, integrating test data generation and APA have become part of the proposed work. This study utilized Pex [51] to generate the desired test data to analyze white-box testing on students' assignment. This tool employed the technique of dynamic symbolic execution that is similar to what has been proposed by Ihantola [49]. Thus, this study also focuses the structural based test data generation technique.

A study by Hakulinen and Malmi [52] has yet appeared to be the latest work. This study utilises Quick Response (QR) code in automated assessment of program correctness and has been supported by a technique of randomized input to generate the required test data. The assessment takes images as input and produces a valid QR code as output when solved correctly. The proposed method provides automated assessment by taking advantage of mobile devices and support for multiple programming languages. The same as the work proposed by Malmi *et al.* [11], the random technique is someway not reliable.

Table 1 summarises the trend of the work described. It shows that none of the studies have proposed both functional and structural based test data generation techniques for assessing students' programming exercises. Since the study on APA mainly aims at producing more accurate markers, an integration of both the techniques appears to be a better solution. It is because both the aspects of functional and structural coverage of a program code are considered as quality factors in marking students' programs. In addition, all the past studies have merely used the techniques that do not fully cover the criteria of reliability and validity of test data adequacy (or the positive and negative testing criteria) to achieve a criterion that is so-called 'complete' test criterion. These criteria ensure programming assessments can be accomplished more efficiently as the derived test data do cover the adequate tests that adhere to the given programming exercise specifications and error-prone point coverage.

Complementing existing work and improving the state-of-the-art in APA, it can be deduced that test data generation plays an important part especially for dynamic testing. A systematic way to automatically derive and generate an adequate set of test data for APA would generally give useful benefit to lecturers of programming courses. Besides, the effort to improve the shortcomings can enhance the features of APA especially for the part of feedbacks received by students' as the final assessment result. Furthermore, sufficient feedback is an integral part in APA to allow students develop their programming skill through learning from unexpected mistakes made by them. It can also improve the means of assessing the quality of students' programs as more accurate.

Table 1 Research trends of the integration of automatic test data generation and programming assessment

Author (s)/year	Testing category		Method		Versatile feature	Quality factor	TDG technique
	BBT	WBT	St	Dy			
Malmi *et al.* [11]	✓			✓	✓	Correctness	Random
Shukur *et al.* [48]	✓			✓	✓	Correctness	Boundary value analysis
Ihantola [49]		✓		✓	NA	Correctness	Symbolic execution with Java PathFinder
Tilmann *et al.* [50]		✓		✓	✓	Correctness	Dynamic symbolic execution with Pex
Hakulinen and Malmi [52]	✓			✓	✓	Correctness	Random

Note: St is *Static*, Dy is *Dynamic*, BBT is *Black-box Testing*, WBT is *White-box Testing*, TDG is *Test Data Generation*, and NA is *Not Available*

Therefore, this study proposes a test data generation framework (or FaSt-Gen) to derive and generate test set and test data that satisfy the above mentioned features. The framework is expected to improve and enhance the features of APAs in giving useful and sufficient feedback to students' work. Furthermore, it could also serve lecturers with a mechanism in which they can furnish an adequate set of test data to be used in assessing students' programs regardless of having the optimal expertise in the particular knowledge of test cases.

3.0 FaSt-Gen AND AUTOMATIC PROGRAMMING ASSESSMENT

FaSt-Gen is a framework to derive a schema of test set that includes an adequate set of test data to perform APA. The framework comprises two parts, which are functional and structural test data generations. Both of them represent the parts that provide test data to implement functional and structural testing respectively as aspects of program quality judged in assessing students' programs. Design of FaSt-Gen is supported by the result of the literature surveys in our previous work [53] and findings of the conducted preliminary study [54]. In overall, the framework of criteria chosen for FaSt-Gen is shown in Figure 1.

Based on Figure 1, the criteria that contribute to FaSt-Gen are based on three viewpoints [22]. Firstly,"*Does the technique require the examining of the internal working?*". If the technique does examine the internal workings such as statements, module

variables and others, it falls under white box testing. Otherwise, the technique falls under black-box testing [55]. FaSt-Gen embeds both techniques.

Secondly, "*How does the technique select the test data?*". The selection of test data depends on the functional (the test data is derived based on the program specifications) or the structural (that depends on the paths of the program or path coverage) of the program. Commonly, for programming exercises the program specifications refer to a list of specification statements that comprise of conditions and/or operations that a program intends to accomplish or satisfy respectively.

Thirdly, "*What type of test data does the technique generate?*". In this study, we select test data according to a certain adopted criteria. The adopted criteria rely on the evaluation criteria that suit the predetermined classification techniques. It involves the issues:

(i) "*When should testing stop?*". Since this study concerns the way of assessing the adequacy of dynamic testing, it depends on the test adequacy criterion [56]. Goodenough and Gerhard [27] defined the concept of an ideal test criterion, which is based on the assumption that the purpose of testing is to determine whether the Software Under Test (SUT) contains any errors.

(ii) "*How testing criterion is selected?*". For the functional test data generation, to compliment the positive testing in deriving test data, we also embed the negative testing which concerns certain error-prone points to provide an ideal test adequacy criterion. For structural test data generation, a path coverage adequacy criterion is chosen as its thoroughness score is 4 out of 5 and it can achieve 100% path coverage [57]. Instead of considering all finite paths from start to end (positive testing), we includes testing for the specific location of errors which might break the path property (negative testing) similar to functional test data generation.

(iii) "*How to design the test set?*". The design of a test set is directly influenced by the chosen adequacy criterion. In designing test cases, apart from inclusion to valid (true) and invalid (false) path conditions, this study also embeds an illegal path condition to cover positive and negative testing criteria. This study proposes an integration of specification-derived test and simplified boundary value analysis to cover the adequacy criteria for functional testing. Due to the fact that, programming exercises have always been driven by the statements of specification hence the most appropriate technique to design the test set is the specification-derived test. In order to ensure it incorporates the negative testing criteria, we integrate the boundary value analysis technique. For the structural test data generation, test cases are derived in such a way that every path is executed at least once as path coverage ensures coverage of all the paths from start to end. In order to obtain a limited number of paths to be covered, this study employs boundary-interior path testing technique [58]. Figure 2 presents the overall view of FaSt-Gen in APA.

Figure 2 shows that both functional and structural test data generations produce a schema of test set separately. The schemas comprise of test data to test students' programs in terms of the correctness of program logic behaviour and its implementation respectively. These test data will be used to verify the correctness level of a program under testing in APA. A lecturer is responsible to prepare programming exercises with their solution models. The students play their roles in preparing and submitting programming solution to each programming exercise. They will be notified with feedback as the final assessment results. Both of the functional and structural test data generation will result in a schema of test set, which includes the adequate test data based on an integration of positive and negative testing criteria and weight values derived from weighted scoring scheme [59].

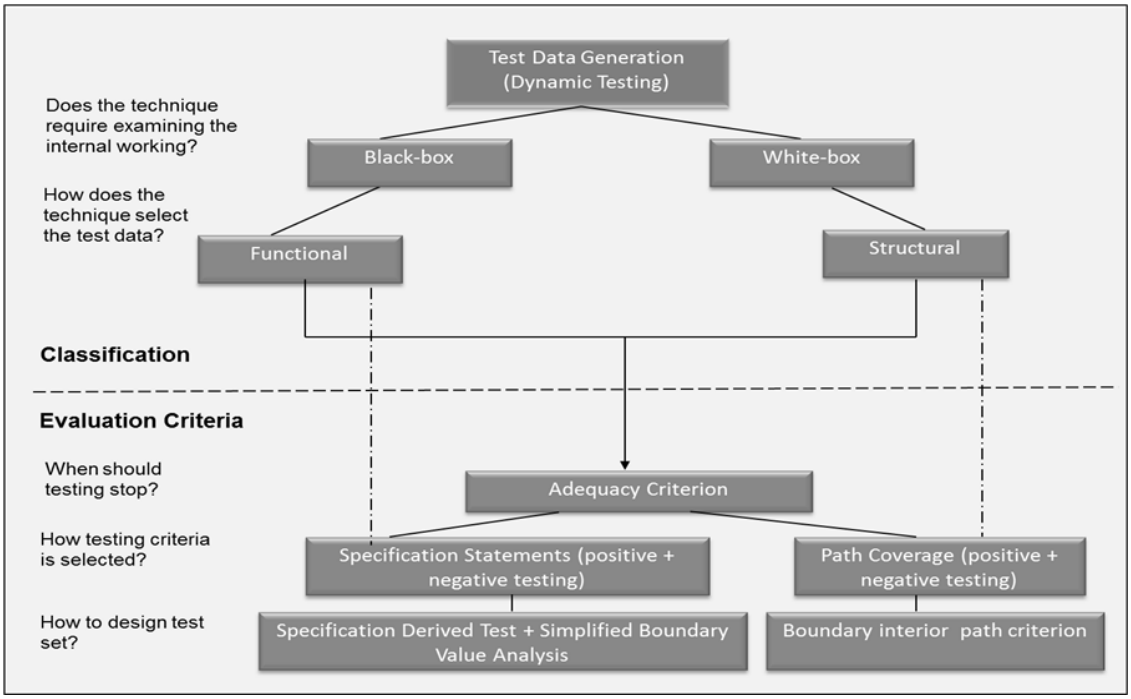


Figure 1 The framework of criteria chosen for FaSt-Gen (adapted from Chu [22])

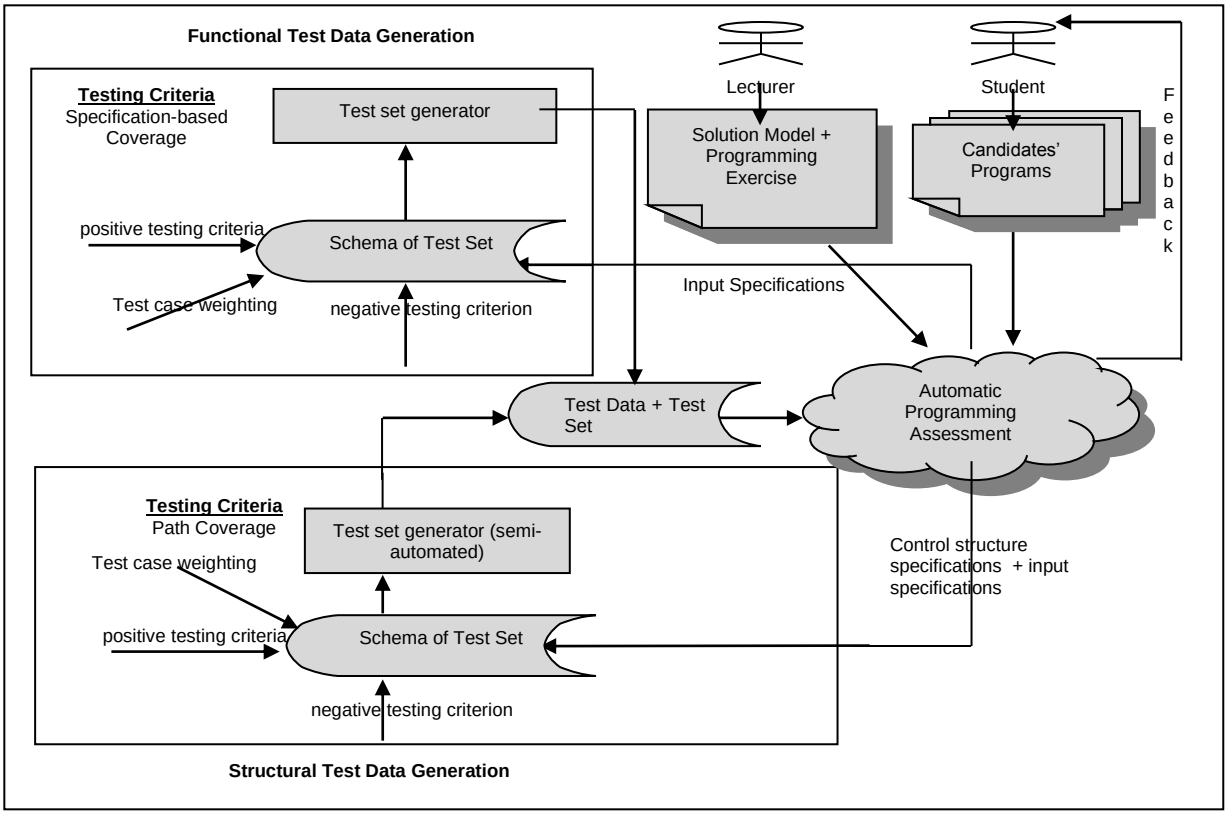


Figure 2 Overall view of FaSt-Gen in APA

4.0 FaSt-Gen: EXAMPLES OF DERIVED TEST SET

In order to map between FaSt-Gen and APA, we include two samples of programming exercises (see Figure 3 and Figure 4). Figure 3 depicts an example of a programming exercise with its specifications and Table 2 tabulates its generated test set (focuses on the functional test data generation). Our previous work [60] has detailed out the design of the functional test data generation. From the table, the total number of test cases is sixteen because both input variables *num1* and *num2* influence output variable *sum* (*testCase*(*num1*) ∪ *testCase*(*num2*) = 4 × 4 = 16). Here, we elaborate the test cases of *TC1* and *TC2* to describe the meaning of the test set as a whole. *TC1* refers to a test case that exercises at the boundaries of valid input partition (V_b) of *num1* and *num2* with their respective values, which are -1 and 4. The values of *num1* and *num2* are valid as long as they are integer numbers and in between the range of [-10, 10]. If the solution to the programming exercise is running against these values, it produces the output Sum = 3, which satisfies the given specification (the value of *sum* is an integer). The test case of *TC2* will exercise at the boundary of valid input partition (V_b) of *num1* and in between the upper and lower boundaries-left of valid input partition ($V_{ulb\text{-}left}$) of *num2*. Their respective values are -14928 and 4. Assume that $V_{ulb\text{-}left}$ of *num2* refers to the valid value of *num2* (in this case is in between the range of [-10,000, -20,000],). If a student's programming solution is executing against the input values, its expected output value is Sum=-14924, which does satisfy the functional specification.

Question:
Write a program that reads two integer numbers, num1 and num2 and prints the total of sum of the numbers.
Functional specification:
Input – Two integer numbers, num1 and num2
Output – Sum (sum = num1 + num2), which is an integer number
Functional process:
- If the inputs are integer numbers, then the total of sum produced as the output is also an integer number.
- Format of program input and output are as follows:

Input:
2
- 5
Output:
Sum = -3

Figure 3 Sample programming exercise and its specifications for numeric global

For the structural test data generation, Figure 4 depicts the sample of programming exercise and its functional specifications, and Figure 5 is its respective flow-graph representation. Table 3 shows the derived test set, which includes test cases that cover valid, invalid and illegal path conditions. The detailed design of the structural test data generation can be found from our previous work [61].

Based on Figure 5, the program produces two linearly independent paths which are; *Path1* → b, a1, a2, e and *Path2* →b, a1, a3, e. For this study, these paths cover the valid path conditions and they are compulsory to be exercised because they are commonly a part of program specifications in APA. The same applies to the path that does not fulfil either *Path1* or *Path2* that is the path of b, a1, e, which cover invalid path conditions. Both valid and invalid path conditions fall into positive testing criterion.

Table 2 Generated schema of test set for the sample programming exercise in Figure 3

Test Case (TC)	Input		Output	Test Case Description
	num1	num2		
TC 1	-1	4	Sum=3	V_b of num1 and num2
TC 2	-14928	4	Sum=-14924	V_b of num1 and $V_{ulb\text{-}left}$ of num2
TC 3	11529	4	Sum=11533	V_b of num1 and $V_{ulb\text{-}right}$ of num2
TC 4	-1	-18620	Sum=-18621	V_b of num1 and IL of num2
TC 5	-14928	-18620	Sum=-33548	$V_{ulb\text{-}left}$ of num1 and V_b of num2
TC 6	11529	-18620	Sum=-7091	$V_{ulb\text{-}left}$ of num1 and $V_{ulb\text{-}left}$ of num2
TC 7	-14928	-18620	Sum=-7091	$V_{ulb\text{-}left}$ of num1 and $V_{ulb\text{-}right}$ of num2
TC 8	hmm	-18620	Null/Error	$V_{ulb\text{-}left}$ of num1 and IL of num2
TC 9	-1	10020	Sum=10019	$V_{ulb\text{-}right}$ of num1 and V_b of num2
TC 10	-14928	10020	Sum=-4908	$V_{ulb\text{-}right}$ of num1 and $V_{ulb\text{-}left}$ of num2
TC 11	11529	10020	Sum=21549	$V_{ulb\text{-}right}$ of num1 and $V_{ulb\text{-}right}$ of num2
TC 12	hmm	10020	Null/Error	$V_{ulb\text{-}right}$ of num1 and IL of num2
TC 13	-1	izp	Null/Error	IL of num1 and V_b of num2
TC 14	-14928	izp	Null/Error	IL of num1 and $V_{ulb\text{-}left}$ of num2
TC 15	izp	11529	Null/Error	IL of num1 and $V_{ulb\text{-}right}$ of num2
TC 16	izp	hmm	Null/Error	IL of num1 and IL of num2

Based on Table 3, the test cases of TC1, TC2 and TC3 represent positive testing criteria (or test data adequacy-reliability) and the test case of TC4 covers negative testing criteria (or test data adequacy-validity). Specifically, TC1 and TC2 cover the valid path conditions and TC3 covers the invalid path condition. As the parameter of *age* is an integer data type, a String value is used to cover the illegal path condition. This condition results the program under testing to return an exception error, which determines

it is the point where an error occurs due to the input-mismatch exception. Considering the flow graph in Figure 5, the illegal path condition can take part as long as the test datum to represent the parameter of *age* is a non-integer data type. Although a number of test cases can cover such path conditions, this study merely selects one value of test datum to represent a single type of exception error. It is mainly to reduce the overall number of test cases generated especially in the case of the number of input variable/parameter increased significantly. It is adequate in APA since it covers the negative testing criteria.

Question:
Write a program that reads an age of a person, which is an integer, and prints the status of the age that is based on the following:

age	status
0 ≤ age ≥ 21	"Youth is a wonderful thing. Enjoy"
age > 2	"Age is a state of mind. Enjoy"

Functional specification:
Input – an age, which is an integer value
Output – status of the age, which is a String
Functional Process:
- If the age is an integer and it fulfils one of the listed condition (0 ≤ age ≥ 21 or age > 21), the program shall return the corresponding status of the age as the program output.
- If the age value is less than zero or a character, the program returns a null value.
- If the age value is a String, the program returns an exception error message.

Figure 4 Sample of programming exercise and its functional specifications

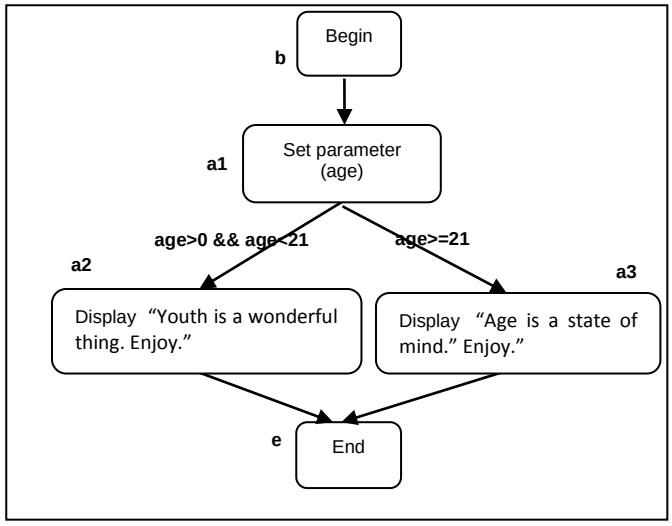


Figure 5 Flow graph that represents the fragment of code shown in Figure 4

Based on Table 3, the test cases of TC1, TC2 and TC3 represent positive testing criteria (or test data adequacy-reliability) and the test case of TC4 covers negative testing criteria (or test data adequacy-validity). Specifically, TC1 and TC2 cover the valid path conditions and TC3 covers the invalid path condition. As the parameter of *age* is an integer data type, a String value is used to cover the illegal path condition. This condition results the program under testing to return an exception error, which determines it is the point where an error occurs due to the input-mismatch exception. Considering the flow graph in Figure 5, the illegal path condition can take part as long as the test datum to represent the parameter of *age* is a non-integer data type. Although a number of test cases can cover such path conditions, this study merely selects one value of test datum to represent a single type of exception error. It is mainly to reduce the overall number of test cases generated especially in the case of the number of input variable/parameter increased significantly. It is adequate in APA since it covers the negative testing criteria.

Table 3 Schema of test set for the fragment of code in Figure 4

Test Case (TC)	Input *age*	Test Case Description	Path Covered	Path Condition
TC 1	18	Exercise the branch of *age*>0 && *age*<21	b, a1, a2, e	Valid branch of *age*>0 && *age*<21
TC 2	45	Exercise the branch of *age*>21	b, a1, a3, e	Valid branch of *age*>21
TC 3	-4	null	b, a1, e	Invalid branches of *age*>0 && *age*<21and Invalid *age*>21
TC 4	"abc"	Error and the program terminate	None	Illegal path

5.0 ANALYSIS AND RESULTS

In order to measure the completeness coverage of the test data adequacy of FaSt-Gen in terms of the selected criteria that are reliability and validity test data adequacy, we conducted a controlled experiment that employs the one-group *pretest-posttest* design [62]. This study also conducted a comparative study evaluation to contribute to the means of measuring the completeness coverage of the criteria of test data adequacy of FaSt-Gen inclusion to the findings from the controlled experiment (see Section 5.3).

Figure 6 depicts the design of the experiment. Based on the figure, symbol *X* represents the exposure of the group to the treatment of interest (independent variable), while *O* refers to the measurement of dependent variable. The *pre-test* experiment intends to measure the degree of the completeness coverage of the criteria of test data adequacy (reliability and validity) for *Current Method*

in preparing a set of test data to perform the dynamic testing in programming assessment. The *Current Method* refers to the means of preparing test data based on the individual user's knowledge in a certain test case design.

O (*Pre-test*)	*X* (Treatment)	*O* (*Post-test*)
The criteria of reliability and validity to measure the completeness coverage of test data adequacy of functional and structural TDG in programming assessment (Current Method)	FaSt-Gen (FaSt-generator)	The criteria of reliability and validity to measure the completeness coverage of test data adequacy of functional and structural TDG in programming assessment

Figure 6 Design of the controlled experiment (adapted from [62])

This experiment used three samples of programming exercises as assignments in the scenario setting that each subject should follow. The exercises cover the three main control structures in Java programming, which are sequential, selection and repetition (loop). They are the most important concepts of programming that every student taking elementary of programming course should master. The subjects of the controlled experiment were lecturers who have been teaching the elementary programming course in one of the public universities in Malaysia. There were twelve (12) subjects who at least have been teaching the programming course for one semester. Due to the logistic constraints such as the difficulty to access subjects of the experiment who were located across different geographies and the needs to have a tightly control setting, only one university was selected to conduct the experiment. Besides, the syllabi of elementary programming courses are almost similar across universities in terms of the topics covered. Thus, it is impractical to conduct a series of experiments among lecturers of elementary programming course at different higher learning institutions.

We used a set of *pre-test* and *post-test* questions that consisted of the same content. This section only focuses the analysis and findings of items in Section A of the questions. All items in Section A are close-ended. We included both the criteria of positive and negative testing. We conducted hypothesis testing by running Wilcoxon signed-rank test to test the formulated hypothesis as follows:

(i) Hypothesis I:
Hypothesis **H_o**: FaSt-Gen (*Y*) tends to be smaller than the Current Method (*X*) in terms of the completeness coverage of the criteria of test data reliability in programming assessment, or $U+ = \#(Y < X)$
Hypothesis **H_1**: FaSt-Gen (*Y*) tends to be larger than the Current Method (*X*) in terms of the completeness coverage of the criteria of test data reliability in programming assessment, or $U- = \#(Y > X)$

(ii) Hypothesis II:
Hypothesis **H_o**: FaSt-Gen (*Y*) tends to be smaller than the Current Method (*X*) in terms of the completeness coverage of the criteria of test data validity in programming assessment, or $U+ = \#(Y < X)$
Hypothesis **H_1**: FaSt-Gen (*Y*) tends to be larger than the Current Method (*X*) in terms of the completeness coverage of the criteria of test data validity in programming assessment, or $U- = \#(Y > X)$

Hypothesis I tests the variable of *reliability* for the functional and structural test data generation. The same applies to Hypothesis II, which is with regard to the variable of *validity*. The following sub-sections reveal the analysis and findings for both hypothesis testing.

5.1 Hypothesis Testing I

Table 4 gives the means and standard deviations of the scores obtained under the criteria of test data reliability of Current Method (*X*) and the criteria of test data reliability of FaSt-Gen (*Y*). *X* refers to the mean of deriving test data based on the lecturer's knowledge in a certain test cases design. It is shown that the score obtained by *Y* (*mean* = 3.00) is larger than *X* (*mean* = 1.9167). However, *X* showed greater variability of scores (*s.d.* = 0.26827) as compared to *Y* (*s.d.* = 0). The mean and standard deviation do not show any statistical difference between the two samples. The standard deviation of the score obtained under the criteria of test data reliability of *Y* is zero because *Y* appears to cover the criteria of reliable test data adequacy in all the three samples of programming exercises used in the experiment. As *Y* is realized as a tool and all subjects referred to the same statements of specifications, the same outputs were produced.

Table 5 provides information about the rank scores. It notifies the number of negative ranks and the number of positive ranks. From the figure, it shows that for all subjects, their score on *Y* was greater than *X*. There were no tied ranks because no subject scored the same for both *Y* and *X*. The footnotes below the ranks table shows how the positive and negative ranks relate.

The Wilcoxon test (see Table 6) evaluated the difference between medians for the score of *Y* and *X* concern, is significant $z = -3.165$, $p < (0.002/2)=0.001$. The findings indicate significant differential concern for *Y* versus *X*, with the higher coverage of the criteria of reliability test data in *Y*. The mean rank of *Y* is 6.5, while the mean rank of *X* is 0.0. Thus, this finding indicates that there was enough evidence to reject H_0 ($p > .025$).

Table 4 Means and standard deviations of the scores obtained under the criteria of test data reliability of *X* and *Y*

	N	Mean	Std. Deviation	Minimum	Maximum
Test Data Reliability (*X*)	12	1.9167	.26827	1.75	2.50
Test Data Reliability (*Y*)	12	3.0000	.00000	3.00	3.00

Table 5 Table of rank scores between the criteria of test data reliability of *X* and Y

		N	Mean Rank	Sum of Ranks
Test Data Reliability (*Y*) – Test Data Reliability (*X*)	Negative Ranks	0[a]	.00	.00
	Positive Ranks	12[b]	6.50	78.00
	Ties	0[c]		
	Total	12		

a. Test Data Reliability (*Y*) < Test Data Reliability (*X*)
b. Test Data Reliability (*Y*) > Test Data Reliability (*X*)
c. Test Data Reliability (*Y*) = Test Data Reliability (*X*)

Table 6 Table of test statistics for the criteria of reliability test data adequacy

	Test Data Reliability (*Y*) – Test Data Reliability (*X*)
Z	-3.165[b]
Asymp. Sig. (2-tailed)	.002

a. Wilcoxon Signed Ranks Test
b. Based on negative ranks.

5.2 Hypothesis Testing II

Table 7 gives the mean and standard deviation of the scores obtained under the criteria of test data validity of Current Method (*X*) and the criteria of test data validity of FaSt-Gen (*Y*). It is shown that the score obtained by *Y* (*mean* = 3.00) is larger than *X* (*mean* = 1.7083). However, *X* showed greater variability of scores (*s.d.* = 0.49810) compared to *Y* (*s.d.* = 0). Means and standard deviations do not show any statistical difference between the two samples. The standard deviation of the score obtained under the criteria of test data validity of *Y* is also zero due to the same reason as explained for Hypothesis Testing I.

Table 8 provides the information on the rank scores and the findings, which appear exactly the same as the testing for Hypothesis I. The Wilcoxon test (see Table 9) evaluated the difference between medians for the score of *Y* and *X* concerns, is significant z = -3.108, $p < (0.002/2)=0.001$. The findings indicate significant differential concern for *Y* versus *X*, with the higher coverage of the criteria of validity test data in *Y*. The mean rank of *Y* is 6.5, while the mean rank of *X* is 0.0. Thus, this finding indicates that there was enough evidence to reject H_0 ($p > .025$).

Table 7 Means and standard deviations of the scores obtained under the criteria of test data validity of *X* and *Y*

	N	Mean	Std. Deviation	Minimum	Maximum
Test Data Validity (*X*)	12	1.7083	.49810	1.00	2.50
Test Data Validity (*Y*)	12	3.0000	.00000	3.00	3.00

Table 8 Table of rank scores between the criteria of test data validity of *X* and Y

Table 9 Table of test statistics for the criteria of validity test data adequacy

	Test Data Validity (*Y*) – Test Data Validity (*X*)
Z	-3.108[b]
Asymp. Sig. (2-tailed)	.002

a. Wilcoxon Signed Ranks Test
b. Based on negative ranks.

Based on the findings of the testing of Hypothesis I and Hypothesis II, it can be concluded that FaSt-Gen significantly improves the criteria of reliable and valid test data adequacy in programming assessment as compared to what is employed in the current practise of programming assessment. Therefore, it proves that FaSt-Gen is able to furnish an adequate set of test data to execute functional and structural testing of a program for APA.

		N	Mean Rank	Sum of Ranks
Test Data Validity (*Y*) – Test Data Validity (*X*)	Negative Ranks	0[a]	.00	.00
	Positive Ranks	12[b]	6.50	78.00
	Ties	0[c]		
	Total	12		

a. Test Data Validity (*Y*) < Test Data Validity (*X*)
b. Test Data Validity (*Y*) > Test Data Validity (*X*)
c. Test Data Validity (*Y*) = Test Data Validity (*X*)

5.3 Qualitative Evaluation (Comparative Study)

This study also conducted a comparative study evaluation to support the results of the controlled experiment discussed in Section 5.1 and 5.2. It is to compare in terms of the coverage of test data and test cases between FaSt-Gen and the techniques that were proposed by Shukur *et al.* [48], Ihantola [49], and Tillmann *et al.* [50]. These studies are among the relevant benchmarks available to be compared

with FaSt-Gen. The analysis of the comparative study will be shown and discussed separately based on the testing techniques considered (functional and structural testing). Two samples of programming exercises were used to analyse the functional and structural testing separately.

If it is with regard to the functional testing (**Case 1**), the analysis is between FaSt-Gen and the technique proposed by Shukur *et al.* [48]. Otherwise, this study compared FaSt-Gen and the technique proposed by Ihantola [49] and Tillmann *et al.* [50] if it puts emphasis on the structural testing (**Case 2**). The following sub-sections present separately the two cases of the comparative study and the one with regard to the measurement of correctness quality of students' programs in APA.

5.3.1 Test Data Adequacy Criteria of Functional Testing–Case 1

Case 1 is an analysis of comparative study between the technique proposed by Shukur *et al.* [48] and FaSt-Gen. Figure 7 depicts the sample of programming exercise used in the analysis of **Case 1**. Table 10 tabulates the result of analysis of **Case 1**.

Question:
Write a program that reads an integer positive number, num1 and returns a square root of the number.

Functional specification:
Input – an integer positive number, num1
Output – squareRoot, which is an integer number
Functional process:
- If the input is an integer number and greater than 0, then the positive square root of the input shall be returned.
- When given an input of less than 0, the program will return an error message "the value should be greater than zero".
- Format of program input and output as follows:

Input:
4
Output:
2

Figure 7 Sample programming exercise used in analysis of Case 1

As shown in the table, the work proposed by Shukur *et al.* [48] produces more in the number of test cases compared to FaSt-Gen, in terms of the test data adequacy (reliable and valid). The study appears to not include the full coverage of valid test adequacy criteria (negative testing). Even though BVA technique does cover negative testing criteria [63], the study seems to exclude the test data that lead the program under testing to return an error of exception. In addition, in terms of the range of values generated to represent test data, they do not cover a wide range of values and seem likely among the static values. However, FaSt-Gen covers a rather wide range of values depending on the maximum and minimum data type ranges of input variables involved. Thus, from this result, it can be concluded that FaSt-Gen furnishes a more adequate set of test data compared to the technique proposed by Shukur *et al.* [48].

5.3.2 Test Data Adequacy Criteria of Structural Testing–Case 2

Case 2 is an analysis between the technique proposed by Ihantola [49], and Tillmann *et al.* [50] and FaSt-Gen. This study used the sample programming exercise as shown in Figure 8 for the analysis in terms of test data adequacy for structural testing. As it is about structural testing, the program to be used in testing should employ any non- sequential control structures, such as selection or repetition. In this analysis, the research in this thesis used a sample programming exercise that applies the concept of selection.

Table 10 Analysis result of Case 1

Approach or technique / Comparison criteria	The work proposed by Shukur *et al.* [48]	FaSt-Gen
The total number of test cases generated	Five (5)	Four (4)
Technique to derive test data	BVA	Specification derived test + simplified version of BVA
Test data coverage (individual input variable/parameter)	-Test datum at the boundary of input domain, 0 -Test datum just less than 0 (-a), -1 -Test datum just greater than 0 (+a), +1. -Test datum of the most negative integer number (-∞), -10 -Test datum of the most positive integer number (+∞), +10	-Test datum at the boundary of valid input partition, a random value between [0,10] - Test datum between the upper and lower boundaries of valid input partition, a random value between [10,000, 20,000] - Test datum between the upper and lower boundaries of invalid input partition, a random value between [0, -20,000] - Test datum at an error prone point (illegal)
The range of values among test data	Among the small values of integer numbers	A different variety of integer values within the range of values of integer data type

Table 11 shows the result of analysis of **Case 2**. It shows that, in terms of the number of test cases covered, FaSt-Gen is leading the work by Ihantola [49] and Tillmann *et al.* [50] with having extra one test case. Focusing the test data coverage, FaSt-Gen covers all the branches (positive testing criteria) as in the work by Ihantola. However, our approach has an inclusion of test datum to cover the illegal path condition (negative testing criteria). This concludes that FaSt-Gen outperforms the technique proposed by Ihantola [49] and Tillmann *et al.* [50] in terms of test data adequacy criteria for structural testing. If considering the means of generating test data, the work by Ihantola [49] can make full automation to it. In our study, it needs a partial human involvement to assign the data to fit the respective test cases generated (automated generation).

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

personAge	**votingStatus**
1 – 20	"you are not eligible for voting"
≥ 21	"you are eligible for voting"

Functional specification:
Input – personAge is an integer
Output – votingStatus, which is a String
Functional process:
- If the input value of personAge 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 shall return a message "the input is out of the range".
- Format of program input and output are as follows:

Input:
24
Output:
You are eligible for voting

Figure 8 Sample programming exercise used in analysis of Case 2

6.0 CONCLUSION AND FUTURE WORK

This paper has presented a framework of test data generation particularly to execute the functional and structural testing of a program in APA that is called FaSt-Gen. In order to furnish an adequate set of test data to conform to specifications of a solution model as well as to include certain extend of error-prone points coverage, FaSt-Gen embeds the positive and negative testing criteria. This framework is expected to assist the lecturers of the elementary programming course to furnish an adequate set of test data to assess the quality of correctness of students' programming exercises both in terms of the dynamic functional and structural testing. It also provides a systematic and consistent way of deriving test data among different individual lecturers regardless of necessarily having an optimal expertise in the knowledge of test cases design. In addition, this feature benefits the lecturers in terms of reducing the course workloads. These include the time spent for marking programming exercises and avoiding them to consider the critical aspects of designing the suitable test cases to judge the correctness quality of students' programs.

The examples included in this paper show that the derived test set and test data do fulfill the criteria of an ideal test criterion that is both reliable and valid [27]. Also, based on the findings collected from the conducted experiment and comparative study evaluation as discussed earlier, it can be deduced that FaSt-Gen significantly improves the criteria of reliability and validity test data adequacy compared to as employed in the current practise of programming assessment. Thus, it can be said FaSt-Gen is able to furnish an adequate set of test data as it covers what have been applied is the practice of programming assessments.

Table 11 Analysis result of Case 2

Approach or technique / Comparison criteria	**The work proposed by Ihantola [49]**	**The work proposed by Tillmann *et al.* [50]**	**FaSt-Gen**
The total number of test cases covered	Four (4)	Four (4)	Five (5)
Technique to derive test data	Symbolic execution with Java Pathfinder (path coverage)	Dynamic symbolic execution with Pex (path coverage) based on parameterized unit testing	Path coverage based on control flow analysis (integrate positive and negative testing criteria + boundary-interior path testing)
Test data coverage	- Test datum at the branch of 0 ≤ personAge ≤ 20. - Test datum at the branch of personAge < 0. - Test datum at the branch of 20 < personAge ≤ 200. - Test datum at the	- Test datum at the branch of 0 ≤ personAge ≤ 20. - Test datum at the branch of personAge < 0. - Test datum at the branch of 20 < personAge ≤ 200. - Test datum at the branch of personAge > 200.	- Test datum at valid branch of 0 ≤ personAge ≤ 20 - Test datum at valid branch of 20 < personAge ≤ 200 -Test datum at invalid branch of 0 ≤ personAge ≤ 20 -Test datum at invalid

	branch of personAge > 200.		branch of 20 < personAge ≤ 200 - Test datum to cover illegal path condition.
Values of test data	The values are randomly generated based on the path condition of certain branch is true.	The values are generated through input parameter based on the path condition of certain branch is true.	Need a human involvement (lecturer) to assign test data values to each of the respective test case generated.

As for the future work, in order to realize the approach so that it can be generalized and applicable in the context of software testing research area (particularly for a large scale of testing), an application of any meta-heuristic algorithm or a hybrid among of them possibly becomes a better solution. In our previous work [53], it shows that meta-heuristic algorithms [64] have become the popular approach applied since early 2000. The main reason is definitely due to the most optimum set of test data as one of the way to ensure the testing process can be undertaken efficiently as one of the main issues and challenges in the area of automated test data generation is the exposure to the NP-hard or non-deterministic polynomial hard problem (the time complexity of $O(n^n)$). Other latest meta-heuristic algorithms such as Harmony Search, and/or Fire-fly could become a promising alternative as well.

In terms of the final output of the assessment, the proposed framework does not provide explicitly the description on the suggestions of what are necessary to be taken by students to improve their programs if their programs are fully or partially incorrect. This additional feature able to assist students to learn from their mistakes to produce a better quality solution to programming exercises. Possibly, one of the resolution strategies is by integrating the data mining concepts focusing the classification methods (intelligent or non-intelligent methods). The set of rules (or training data) might refer to the collections of possible expected outputs (correct and incorrect outputs) from testing. Meanwhile, the possible decisions to the rules could be the descriptions to the produced outputs and the suggestions to improve the tested program especially for the incorrect outputs (invalid or error of output).

To wrap-up, the proposed FaSt-Gen is expected to be able to furnish the adequate set of test data to be used to perform the dynamic functional and structural testing of a program executed for APA. This study also anticipates the growing of this kind of tool to support both lecturers and students in managing programming assessments particularly in the context of Malaysia as nowadays only a very small universities have made used of it.

Acknowledgement

The authors acknowledge Fundamental Research Grant Scheme (FGRS) under the Ministry of Education, Cost Centre 4F263 that partially supports this work.

References

[1] Truong, N., P. Bancroft, and P. Roe. 2005. Learning to Program Through the Web. *ACM SIGCSE Bulletin*. 37(3): 9-13.

[2] Shaffer, S. C. 2005. Ludwig: An Online Programming Tutoring and Assessment System. *ACM SIGCSE Bulletin*. 37(2): 56-60.

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

[4] Tremblay, G. and E. Labonte. 2003. Semi-Automatic Marking of Java Programs using Junit. In *Proceeding of International Conference on Education and Information Systems: Technologies and Applications (EISTA '03)*. 42-47.

[5] Saikkonen, R., L. Malmi, and A. Korhonen. 2001. Fully Automatic Assessment of Programming Exercises. *ACM SIGCSE Bulletin*. 33 (3): 133-136

[6] Aleman, J. L. F. 2011. Automated Assessment in Programming Tools Course. *IEEE Transactions on Education*. 54(4): 576-581.

[7] Ihantola, P., T. Ahoniemi, and V. Karavirt. 2010. Review of Recent Systems for Automatic Assessment of Programming Assignments. In *Proceedings of the 10th Koli Calling International Conference on Computing Education Research (Koli Calling '10)*. 86-93.

[8] Jackson, D. and M. Usher. 1997. Grading Student Programs using ASSYST. In *Proceedings of the 28th SIGCSE Technical Symposium on Computer Science Education*. 35-339.

[9] Luck, M. and M. S. Joy. 1999. Secure On-line Submission System. *Journal of Software–Practise and Experience*. 29(8): 721-740.

[10] Blumenstein, M., S. Green, A. Nguyen and V. Muthukkumarasamy. 2004. GAME: A Generic Automated Marking Environment for Programming Assessment. In *Proceedings of the International Conference on Information Technology: Coding and Computing (ITCC'04)*. 2: 212-216.

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

[12] Choy, M., U. Nazir, C. K. Poon and Y. T. Yu. 2005. Experiences in Using an Automated System for Improving Students' of Computer Programming. Lecture Notes in Computer Science Learning (Springer Berlin/ Heidelberg). 267-272.

[13] Higgins, C. A., G. Gray, P. Symeonidis, and A. Tsintsifas. 2006. Automated Assessment and Experiences of Teaching Programming. *Journal of Educational Resources in Computing*. 5(3): Article 5.

[14] Gotel, O., C. Scharff and A. Wildenberg. 2007. Extending and Contributing to an Open Source Web-Based System for the Assessment of Programming Problems. In *Proceedings of the 5th International Symposium on Principles and Practice of Programming in Java (PPPJ'07)*. Lisboa, Portugal. 3-12.

[15] Auffarth, B., M. Lopez-Sanchez, J. C. Miralles, and A. Puig. 2008. System for Automated Assistance in Correction of Programming Exercises (SAC). In *Proceedings of the fifth*

CIDUI–V International Congress of University Teaching and Innovation.
[16] Tremblay, G., F. Gu´erin, A. Pons, and A. Salah. 2008. Oto, A Generic and Extensible Tool for Marking Programming Assignments. *Journal of Software-Practice and Experience*. 38(3): 307-333
[17] Nunome, A., H. Hirata, M. Fukuzawa and K. Shibayama. 2010. Development of an E-learning Back-end System for Code Assessment in Elementary Programming Practice. In *Proceeding of the 38th Annual Fall Conference on SIGUCCS*. Norfolk, VA, USA. 181-186.
[18] Queiros, R. and J. S. Leal. 2012. PETCHA-A Programming Exercises Teaching Assistant. In *Proceeding of the 17th ACM Annual Conference on Innovation and Technology in Computer Science Education (ITiCSE'12)*. Haifa, Israel. 192-197.
[19] Shamsi, F. A. and A. Elnagar. 2012. An Intelligent Assessment Tool for Students' Java Submissions in Introductory Programming Courses. *Journal of Intelligent Learning Systems and Applications*. 4(1): 59-69.
[20] Sherman, M., S. Bassil, D. Lipman, N. Tuck and F. Martin. 2013. Impact of Auto-Grading on an Introductory Computing Course. *Journal of Computing Sciences in Colleges*. 28(6): 69-75.
[21] Malmi, L., R. Saikkonen and A. Korhonen. 2002. Experiences in Automatic Assessment on Mass Courses and Issues for Designing Virtual Courses. In *Proceedings of the 7th Annual Conference on Innovation and Technology in Computer Science Education (ITiCSE' 02)*. Aarhus Denmark. 55-59.
[22] Chu, H. D., J. E. Dobson and I. C. Liu.1997. FAST: A Framework for Automating Statistical-based Testing. *Software Quality Journal*. 6(1): 13-36.
[23] Burnstein, I. 2003. *Practical Software Testing*. New York: Springer-Verlag.
[24] Deimel-Jr., L. E. and B. A. Clarkson.1978. The TODISK-WATLOAD System: A Convenient Tool for Evaluating Student Programs. *Proceeding of 16th ACM Annual Southeast Regional Conference*. Atlanta). 168-171.
[25] Van-Vliet, H. 2008. *Sofware Engineering: Principles and Practice*. 3rd Edition. Great Britain: John Wiley & Sons, Ltd, Glasgow.
[26] Sommerville, I. 1995. *Software Engineering*. 5nd Edition. USA: Pearson-Addison Wesley.
[27] Goodenough, J. B. and S. L. Gerhart. 1975. Towards a Theory of Test Data Selection. *In Proceedings of the International Conference on Reliable Software*. New York, USA. 493-510.
[28] Jackson, D. 2000. A Semi-Automated Approach to Online Assessment. *Proceedings of the 5th annual SIGCSE/SIGCUE Conference on Innovation and Technology in Computer Science Education (ITiCSE '00)*. Helsinki, Finland. 164-167.
[29] Clarke, L. A. 1976. A System To Generate Test Data and Symbolically Execute Programs. *IEEE Transaction on Software Engineering*. SE-2(3): 215-222.
[30] Gupta, N., A. P. Mathur and M. L. Soffa. 1998. Automated Test Data Generation Using an Iterative Relaxation Method. *ACM SIGSOFT Software Engineering Notes*. 23(6): 231-245.
[31] Pargas, R. P., M. J. Harrold and R. R. Peck. 1999. Test-Data Generation Using Genetic Algorithms. *Journal of Software Testing, Verification and Reliability*. 9(4): 263-282.
[32] Offutt, J., S. Liu, A. Abdurazik and P. Ammann. 2003. Generating Test Data from State-Based Specifications. *Software Testing, Verification and Reliability*. 13: 25-53.
[33] Zamli, K. Z., N. A. M. Isa, M. F. J. Klaib and S. N. Azizan. 2007. Tool for Automated Test Data Generation (and Execution) Based on Combinatorial Approach. *International Journal of Software Engineering and Its Applications*. 1(1): 19-36.
[34] Alshraideh, M., L. Bottaci and B. A. Mahafzah. 2010. Using Program Data-state Scarcity to Guide Automatic Test Data Generation. *Software Quality Journal*. 18(1): 109-144.
[35] Zhang, Y., D, Gong and Y. Luo. 2011. Evolutionary Generation of Test Data for Path Coverage with Faults Detection. In *Proceeding of the 2011 Seventh International Conference on Natural Computation (ICNC)*. 4: 2086-2090.
[36] McMinn, P., M. Harman, K. Lakhotia, Y. Hassoun and J. Wegener. 2012. Input Domain Reduction through Irrelevant Variable Removal and Its Effect on Local, Global, and Hybrid Search-Based Structural Test Data Generation. *IEEE Transactions on Software Engineering*. 38(2): 453-477.
[37] Benouhiba, T. and W. Zidoune. 2012. Targeted Adequacy Criteria for Search-based Test Data Generation. In *Proceeding of 2012 International Conference on Information Technology and e-Services (ICITeS'12)*. Sousse, Tunisia. 1-6.
[38] Bhasin, H., N. Singla, and S. Sharma. 2013. Cellular Automata Based Test Data Generation. *ACM SIGSOFT Software Engineering Notes*. 38(4): 1-7.
[39] Monpratarnchai, S., S. Fujiwara, A. Katayama, and T. Uehara. 2014. Automated Testing for Java Programs using JPF-based Test Case Generation. *ACM SIGSOFT Software Engineering Notes*. 39 (1): 1-5.
[40] Guo, M., T. Chai and K. Qian. 2010. Design of Online Runtime and Testing Environment for Instant Java Programming Assessment. In *Proceeding of 7th International Conference on Information Technology: New Generation (ITNG 2010)*. Las Vegas, NV. 1102-1106.
[41] Cheng, Z., R. Monahan and A. Mooney. 2011. nExaminer: A Semi-automated Computer Programming Assignment Assessment Framework for Moodle. In *Proceedings of International Conference on Engaging Pedagogy 2011 (ICEP11)*. NCI, Dublin, Ireland. 1-12.
[42] Jones, E. L. 2001. Grading Student Programs- A Software Testing Approach. *Journal of Computing Sciences in Colleges*. 16(2): 185-192.
[43] Isong, J. 2001. Developing An Automated Program Checker. *Journal of Computing Sciences in Colleges*. 16(3): 218-224.
[44] Edwards, S. H. 2003. Improving Student Performance by Evaluating How Well Student Test Their Own Programs. *Journal on Educational Resources in Computing (JERIC)*. 3(3): 1-24.
[45] Fischer, G. and J. W. Gudenberg. 2006. Improving the Quality of Programming Education by Online Assessment. *Proceedings of the 4th International Symposium on Principles and Practice of programming in Java*. Mannheim, Germany. 208-211.
[46] Rossling, G. and S. Hartte. 2008. WebTask: Online Programming Exercises Made Easy. *Proceedings of ITiCSE'08*. Madrid, Spain. 363.
[47] Jurado, F., M. Redondo and M. Ortega. 2012. Using Fuzzy Logic Applied to Software Metrics and Test Cases to Assess Programming Assignments and Give Advice. *Journal of Network and Computer Applications*. 35(2): 695-712.
[48] Shukur, Z., R. Romli and A. B. Hamdan. 2005. Skema Penjanaan Data dan Pemberat Ujian Berasaskan Kaedah Analisis Nilai Sempadan (A Schema of Generating Test Data and Test Weight Based on Boundary Value Analysis Technique), *Technology Journal*. 42(D): 23-40.
[49] Ihantola, P. 2006. Automatic Test Data Generation for Programming Exercises with Symbolic Execution and Java PathFinder. Master Thesis of Helsinki University of Technology, Finland.
[50] Tillmann, N., J. D. Halleux, T. Xie, S. Gulwani and J. Bishop. 2013. Teaching and Learning Programming and Software Engineering via Interactive Gaming. In *Proceedings of the 2013 International Conference on Software Engineering (ICSE'13)*. San Francisco, CA, USA. 1117-1126.
[51] Tillmann, N. and J. D. Halleux. 2008. Pex-white Box Test Generation for .NET, Tests and Proofs. Lecture Notes in Computer Science. 4966: 134-153.
[52] Hakulinen, L. and L. Malmi. 2014. QR Code Programming Tasks with Automated Assessment. *Proceedings of the 2014 conference on Innovation & technology in computer science education (ITiCSE'14)*. Uppsala, Sweden. 177-182.
[53] Romli, R., S. Sulaiman and K. Z. Zamli. 2010. Automatic Programming Assessment and Test Data Generation: A Review on Its Approaches. In *Proceeding of 2010*

International Symposium on Information Technology (ITSim'10). Kuala Lumpur, M'sia. 1186-1192.

[54] Romli, R., S. Sulaiman and K. Z. Zamli. 2011. *Current Practices of Programming Assessment at Higher Learning Institutions.* CCIS 179 (Springer Berlin/Heidelberg). Part 1: 471-485.

[55] Bache, R. and G. Bazzana. 1994. *Software Metrics for Product Assessment*. International Software Quality Assurance Series, Europe: McGraw-Hill.

[56] Tracey, N. G. 2000. A Search-Based Automated Test-Data Generation Framework for Safety Critical Software. PhD Thesis, University of York, UK.

[57] IPL Information Processing Ltd., Structural Coverage Metrics. 1997. [Online] from: http://www.ipl.com/pdf/p0823.pdf. [Accessed on: 10 Feb 2009].

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

[59] Gillies, A. 1992. *Software Quality: Theory and Management.* Boston: Kluwer Academic Publisher.

[60] Romli, R., S. Sulaiman, and K. Z. Zamli. 2011. Test Data Generation in Automatic Programming Assessment: The Design of Test Set Schema for Functional Testing. *Proceeding of 2nd International Conference on Advancements in Computing Technology (ICACT'11)*. Jeju Island, South Korea.1078-1082.

[61] Romli, R., S. Sulaiman, and K. Z. Zamli.2013. Designing a Test Set for Structural Testing in Automatic Programming Assessment. *International Journal of Advances in Soft Computing and Its Application (Special Issues on Application of Soft Computing in Software Engineering)*. 5(3): 41-64.

[62] Fraenkel, J. R. and N. E. Wallen. 2000. How to Design and Evaluate Research in Education, 4th Edition, USA: McGraw-Hill Companies.

[63] Howden, W. E. 1978. An Evaluation of the Effectiveness of Symbolic Testing. *Software-Practice and Experience*. 8: 381-397.

[64] McMinn, P. 2004. Search-based Software Test Data Generation: A Survey. *Software Testing, Verification & Reliability*. 14(2): 105-156.