Design and Implementation of A Tool to Integrate Automated Test Data Generation and Automatic Programming Assessment

Anas Farhan Tajudin and Rohaida Romli

Universiti Utara Malaysia, Malaysia, {email@anasfarhan.com, aida@uum.edu.my}

ABSTRACT

Nowadays, manually assessing students' programming exercises has been identified as among of the toughest tasks to lecturers of programming courses on top of their high routine workloads. Thus, Automatic Programming Assessment (or APA) has turn out to be an alternative method to automatically assess these students' programming exercises effectively. In APA, test data generation process has mainly being a part of performing dynamic testing on students' programming solutions. Diverse of automated test data generation (or ATDG) methods have been researched in past few years. However, these methods have not been sufficiently and systematically adapted by researches in APA. Nonetheless, it appears very limited studies have put in some efforts to realize an integration of APA and ATDG as a complete APA system (or APAS) so as to provide a better quality and precise program testing on students' programs. So as to realize this limitation, this paper presents the design and implementation of an APAS named Automated Java Programming Assessment Tool or Auto-JPAT that integrates a recent proposed ATDG method named DyStruc-TDG (which covers a dynamic-structural testing). As a whole, Auto-JPAT contributes as a means to educators of programming courses to assess students' programming solutions via a dynamic-structural testing regardless of having any sufficient knowledge on design of test cases.

Keywords: Automatic Programming Assessment (APA), Test Data Generation (TDG), dynamic-structural testing.

I INTRODUCTION

In college or university, first year students who are enrolling in Computer Science, IT or Computer Engineering related programmes are compulsory to register a subject named 'Introduction to Programming' or any identical one so as to allow them proceed with the advanced level of programming subjects. Having to learn programming, practical programming exercises and hands on are necessarily required which are commonly provided through computer lab sessions and assignments. During lab sessions, it is not sufficient to acquire any feedback instantly from lecturers as the process requires longer time to deal with a numerous number of students (Tahbidar and Kalita, 2011). It is pivotal for students to learn a basic programming skill in order to advance to the higher levels. This is the main reason why most of the students do not like the idea of being given too many exercises and assignments by their lecturers.

As for grading students' programming assignments, it will mostly consume most of the lecturers' time load and it is burdensome and tiring task (Liang et al., 2009) particularly when a large number of students are currently registered in a class (Tahbidar and Kalita, 2011). Moreover, it significantly increases their workload, especially when marking longer codes. Therefore, Automatic Programming Assessment (or APA) would be solving these issues by helping the lecturers in reducing their workload from marking the assignments submitted by students. Furthermore, APA offers a systematic and consistent means of testing in assessing students' programming assignments. Generally, when students submitted their assignments, an APA tool (or is ideally called APA System: APAS) will automatically assess these assignments based on a schema pre-set by their lecturers and provide an instant assessment result to students. Thus, APAS has been becoming more pivotal in explicitly supporting teaching and learning programming more effectively (Latiu et al., 2012). Lots of APAS were developed and tested in various recent studies (Jackson, 2000; Liang et al., 2009; Blumenstein et al., 2004; Lim et al., 2008; Brause, 2014; Tillmann et al., 2013; Gotel et al., 2007; Alemán, 2011; Auffarth et al., 2008; Sherman et al., 2013; Nunome et al., 2010; Queirós & Leal, 2012; Saikkonen et al., 2001; Al Shamsi & Elnagar, 2012), such as Assyst, BOSS, GAME, TRAKLA2, PASS, ELP, CourseMaster, WeBWorK-JAG, SAC, Oto, ICAS, PETCHA, eGrader, and Bottlenose.

Software testing, which is a foundation of APA (Ihantola et al., 2010), is defined as a technique to discover, measure, and disclose errors happened in a program (Korel, 1990). In order to rate the quality of a program (Romli et al., 2010) produced by a student, the program needs to be tested hence, this is where software testing plays its role. Software

testing can be distinguished in two types which are static testing and dynamic testing. By a definition, static testing is a testing, which is done without executing the program itself (Edvardsson, 1999). In contrast, program execution is required in dynamic testing with test data set. Furthermore, in dynamic testing, its test can be segregated into two variants which are black-box and white-box testing. Based on a recent review (Monpratarnchai et al., 2014), black-box testing, which is also recognized as functional testing, is a testing which only needs the output of the program only without having to access its source code. Meanwhile, white-box testing, which is also recognized as structural testing, requires to access the internal source code of the program, hence test data is a necessity to conduct these tests.

Test data generation in the area of software testing refers to an approach to generate a test data set to fulfil a certain criterion of testing (Kitaya and Inoue, 2016). It is acknowledged that the test data generation plays an influential role in APA in conducting test, yet they are not utilized frequently in recent studies (Rahman, 2007; Reid, 2015; Romli et al., 2015). With the absence of a test data generator, lecturers would have to set up the schema or generate the test data manually to facilitate them in marking students' programming assignments. Thus, we developed an APAS that is called Automated Java Programming Assessment Tool "(or Auto-JPAT) that integrates our recent proposed method namely DyStruc-TDG to allow automated programming assessments that focuses on dynamic-structural testing (utilizing MC/DC and path coverage testing criteria).

The content of the remaining sections are organized as follows: Section II provides a brief discussion on some related works. Section III details out the design of Auto-JPAT. Section 1V reveals on the implementation of Auto-JPAT and finally, Section V concludes the paper.

II RELATED WORK

It has been more than fifty years since 'Automatic Programming Assessment' (APA) made its introduction. Based on the review by Douce et al. (2005), automated test-based assessment systems are grouped into three generations in which can be described from early assessment tools to command-line interface (CLI) or graphical user interface (GUI) distributed systems and later is developed as web-based systems. Later on, Ihantola et al. (2010) also provided a more detailed review of latter modern assessment tools. In their review, they have identified that most of the 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 behaviour of the programs they required for.

APA has been a great help especially to the lecturers who found manually marking and grading the students' assignments is a burden task. Since APA provides instant feedback, the students' learning process will be faster and enhanced effectively. A recent study conducted by Liang et al. (2009) outlined that there were a small number of APA that gave rich, critical and timely response. More details on the advantages of APA has been outlined in a study conducted by Rahman and Nordin (2007). On top of that, a review study conducted by Romli et al. (2014) which focused on the APA trends stated that dynamic testing is the most used testing method in APA. The study also outlined that black-box testing is extensively used in APA with correctness factor as the most popular quality factor since it provides satisfaction to the lecturers while assessing the assignments. In term of test data generation, most of them are done manually instead of automatically generate them. In a summary, the development of APA has been a great platform for lecturers in reducing their time and workload for marking and grading the students' assignments.

In software testing, it is been many years since various techniques for Automated Test Data Generation (ATDG) have been studied. Test data generators were commonly developed as tools to support ATDG. Based on Korel (1990), test data generators are divided into three types: path-wise test data generators, data specification generators, and random test data generators. From the study, the author presented path-wise test data generator using dynamic testing method. On the other survey by Edvardsson (1999), the author presented a test data generator which is consisted of a path selector, a program analyser, and a generator itself. However, it is summarized that the approach is mainly focused on structural testing. Another survey by Romli et al. (2014) on ATDG indicated that the most test data generations are mainly utilized white-box testing techniques, while path coverage is primarily used as the metric for coverage. Another study done by Monpratarnchai et al. (2014) also focused on path coverage metric via the usage of symbolic execution with Java PathFinder (JPF) in generating test data.

APA has been gaining more popularity from researchers in Computer Science eduaction in order to assess students' programming assignments. However, there are still limited studies regarding the integration of APA together with ATDG that focus on dynamic-structural testing. In an earlier

study conducted by Ihantola (2006) utilized a symbolic execution technique with the extension of JPF library to generate test data set. The technique itself is integrated with APA. In another study by Tillman et al. (2013), they also utilized dynamic symbolic execution technique for test data generation. Another recent study by Romli et al. (2015) presented a framework for test data generation that covered both structural and functional testing in which utilized positive and negative testing technique for its test data generation. This study proposed an integration of specification derived test and simplified boundary analysis techniques to derive the required test data. Similarly to the other two studies, this study also focused the criteria of path coverage testing.

III DESIGN OF Auto-JPAT

Auto-JPAT is a web-based application or tool which has basic features or modules including basic content management system and file management system. Since DyStruc-TDG method is integrated with the tool, it also covers basic APAS features. Figure 1 depicts an overall design view of Auto-JPAT. A lecturer is obliged to devise programming exercises with their solution models. The students play their parts by preparing and submitting programming solutions for each programming exercise. They will be notified with instant feedbacks as the final assessment results after Auto-JPAT finished assessing their programming solutions. The process of test data generation will result a schema of test set, which includes a possible set of test data based on positive testing criteria and weight values derived from weighted scale set by the lecturers.

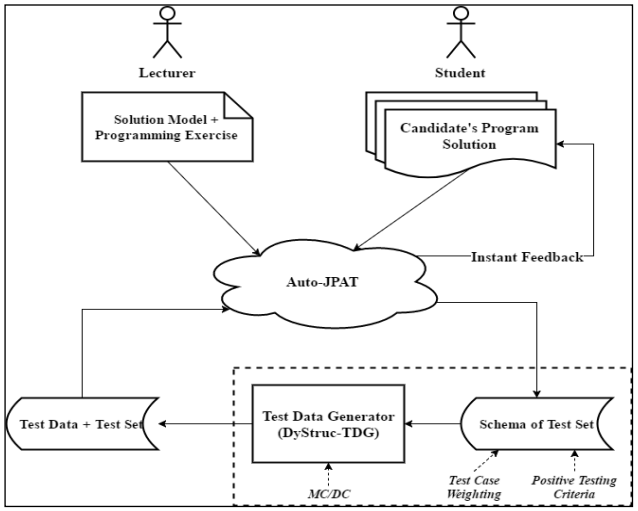


Figure 1. Overall view of Auto-JPAT

Three user group accesses are created for invoking all functions in Auto-JPAT, which are Super Administrator (*SuperAdmin*), lecturers/instructors, and student. The privileges for these group accesses are as follow:

i) *Super Administrator (SuperAdmin)*

SuperAdmin has absolute privileges or authorities to access all administrative features in Auto-JPAT such as user management and system management. For evaluation, *SuperAdmin* is responsible to set up the Auto-JPAT including manage users, manage the course groups, and manage course subjects to ensure the evaluation sessions go smoothly without hitch. These administrative privileges are only given to *SuperAdmin*.

ii) *Lecturer/Instructor*

Lecturer (or Instructor) is given a number of privileges. These privileges are including learning materials (notes) management and assignments management. In term of learning materials management, the lecturer is able to upload (add), update the details (edit), download (view), and remove (delete) the learning materials. While for assignments management, the lecturer can add, edit, view, and delete the assignment. In addition to view the uploaded assignment, the lecturer is able to view the test cases (test data) generated and update the weightage for each test case generated. And also, the lecturer is able to view the list of students' assignment submission.

iii) *Student*

Student is the user group access given with very limited privileges including download (view) the learning materials (notes), and assignment management functions such as view the assignment, upload the assignment solution file, and assess the assignment.

For lecturers, there are eight (8) use cases which are Manage User Profile, Manage Assignment, Create New Assignment, View Assignment, Prepare/Edit Test Data, View Submission List, View Course Planner, and Login use cases. Figure 2 illustrates all use cases for the lecturers. For students, there are four (4) use cases which are Manage User Profile, Manage Assignment, View Course Planner, and Login use cases. Figure 3 illustrates all use cases for the students. Table 1 lists the overall mapping between functions or features in Auto-JPAT and group accesses in Auto-JPAT.

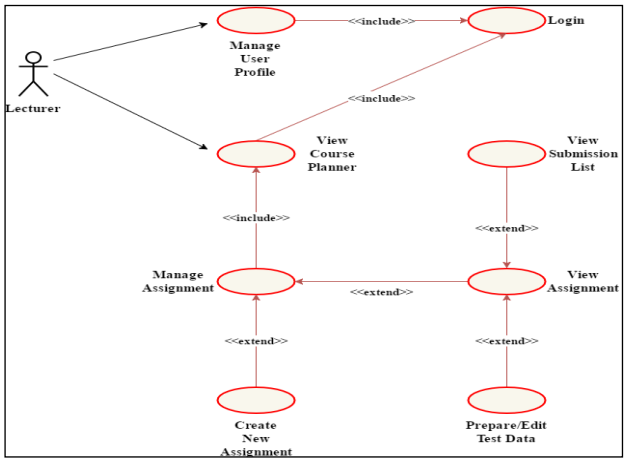


Figure 2. Use Case Diagram for Lecturers

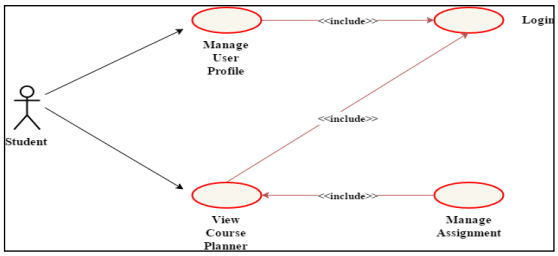


Figure 3. Use Case Diagram for Students

Table 1. Overall Mapping Between Functions and User Group Accesses in Auto-JPAT

Function/Privilege	Group Access		
	SuperAdmin	Lecturer	Student
Academic Session Management - *add/edit/view/delete*	√		
Academic Semester Management - *add/edit/view/delete*	√		
Course Management - *add/edit/view/delete*	√		
Course Group Management - *add/edit/view/delete*	√		
User Management functions - *add/edit/view/delete*	√		
User Profile Management - *edit/view*	√	√	√
View Enrolled Courses		√	√
View Course Planner		√	√
Add New Assignment		√	
View Assignment: Details		√	√
View Assignment: Test Data		√	
View Assignment: Assessment Details			√
View Assignment: Submission List		√	
View Schema Solution File		√	
View Submission Solution File		√	√
Assess Assignment			√
Delete Assignment		√	
Add New Learning Material		√	
View Learning Material		√	√
Delete Learning Material		√	

IV IMPLEMENTATION OF Auto-JPAT

The process of implementing Auto-JPAT was done by adopting an extreme prototyping model. Since Auto-JPAT is a web-based system, the model is suitable to be used as it is applicable for super-fast project cycling and delivery (Dow, 2011). A development of Auto-JPAT also utilized the technology of JavaServer Pages (JSP) and DyStruc-TDG has been integrated as part of the system. Table 2 has listed the related components and software tools used in the implementation of Auto-JPAT.

After the development of Auto-JPAT has been accomplished, the Auto-JPAT had undergone a prototype testing to ensure all functions work properly and smoothly so as the planned user experience evaluation will be conducting with absence of any problem. In case of unwanted bugs and errors discovered during the testing, they were fixed out immediately. Testing for Auto-JPAT was performed with the following setup of hardware and software:

i) Hardware setup - The testing was performed on Windows Operating System (OS) based computer (Windows 10 Pro) that can access Auto-JPAT.

ii) Software setup - The testing was performed with required software components as listed in Table 2.

Table 2. Component Tools Used in Implementation of Auto-JPAT

Component Tool	Purposes
Eclipse IDE Neon.2	A Java EE IDE to develop Auto-JPAT including designing the interfaces for webpages of Auto-JPAT and integrating DyStruc-TDG generator with Auto-JPAT
Java™ SE Development Kit 8 (JDK 8)	A compiler to compile and run Java program (program schema and students' programming solutions
mysql-connector-java-5.1.19-bin.jar	JDBC Driver for MySQL (Connector/J) that translates JDBC calls into standard protocol used by MySQL
jstl-1.2.jar	JavaServer Pages Standard Tag Library (JSTL) which provides easier standard tags for writing JSP codes
javaparser-core-2.3.0.jar	An external Java library used in DyStruc-TDG generator for test data generation.
commons-exec-1.3.jar	An external Java library used to assist the execution of schema solution and students' programming exercises
Java & JavaServer Pages (JSP)	Language or technology behind the interaction of Auto-JPAT webpages
MySQL 5.0.11	An open source database management system which was used to manage Auto-JPAT database
SQLyog Community v12.2.2	A MySQL client for administration and management of Auto-JPAT database
Apache Tomcat 7	A software used to deploy Auto-JPAT tool

For this study, unit testing and integration testing were conducted on Auto-JPAT. Unit testing focuses on testing the functionalities of Auto-JPAT by following step-by-step of procedures listed as a set of test scripts developed for all available functions as listed in Table 1. On the other hand, the integration testing (Leung and White, 1990) focuses on testing all other features and functionalities as well as the integration part that embeds the generator DyStruc-TDG in order to ensure all those features and functionalities work properly in Auto-

JPAT. For this purpose, it involved testing the functionalities of DyStruc-TDG and overall functionalities of Auto-JPAT with respective samples of programming exercises. The sample programming exercises comprise of two simple programming exercises involving selection and repetition (loop) control structurres.. Figure 4 illustrates the overall architecture of Auto-JPAT.

The consecutive Figure 6, Figure 7 and Figure 8 show among the important user graphical interfaces for Auto-JPAT based on the main role of users (lecturer and student). Based on Figure 6, every user either student or lecturer must log into Auto-JPAT with a valid username and password before accessing other features or functions in Auto-JPAT. Dashboard interface is shown each time a user successfully logged into Auto-JPAT. Every Auto-JPAT user is given access to manage his/her user profile settings such as profile picture and account password. This interface can be accessed by selecting 'Profile' from navigation bar menu or clicking 'My Profile' button in the Dashboard interface. Students and lecturers may view the list of courses that have been enrolled in a particular semester via "Enrolled Course List Interface". Course Planner interface will display a course planner for selected course group in form of weekly record.

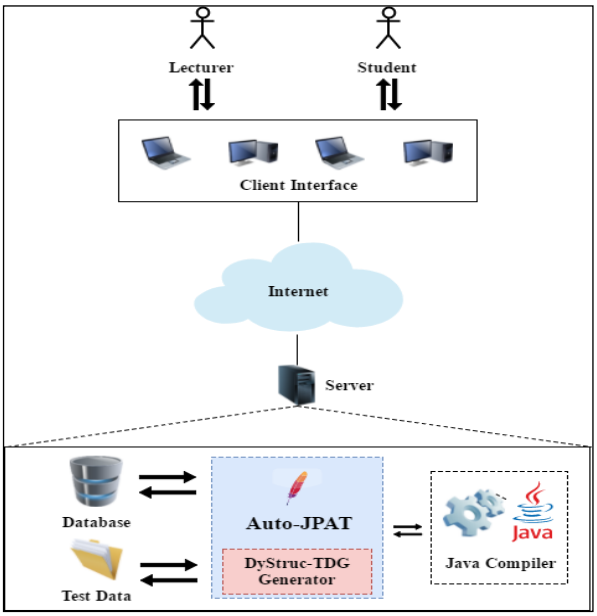
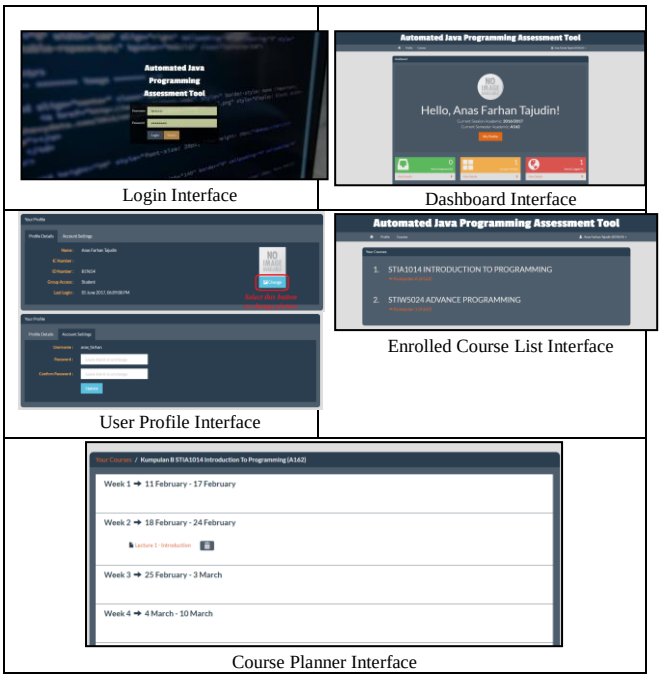


Figure 4. Overall Architecture of Auto-JPAT

Based on Figure 7, it shows the interfaces that students can access. The students only can view the assignment created by the lecturers. While viewing the assignment, they can either submit the assignment solution if they have not yet submitted via 'Upload' button or initiate assessment process via 'Assess' button after they have successfully submitted their solutions. The system does not allow resubmission once assessment process is starting, so they are given a confirmation message either to continue the process or not. Once the assessment was done, the students can see their assessment result in 'Assessment Details' tab.

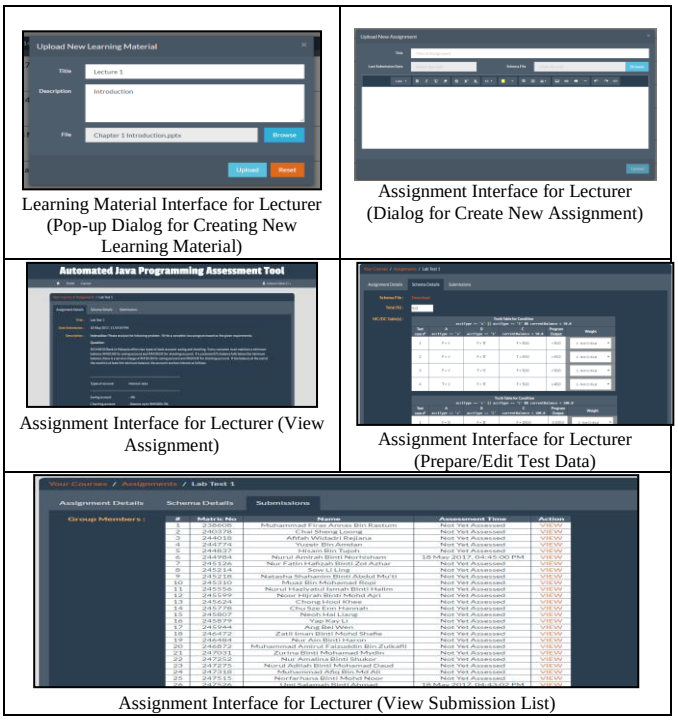


Login Interface | Dashboard Interface

User Profile Interface | Enrolled Course List Interface

Course Planner Interface

Figure 5. General Interfaces for all users



Learning Material Interface for Lecturer (Pop-up Dialog for Creating New Learning Material)

Assignment Interface for Lecturer (Dialog for Create New Assignment)

Assignment Interface for Lecturer (View Assignment)

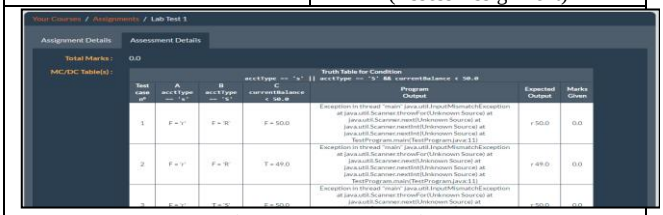
Assignment Interface for Lecturer (Prepare/Edit Test Data)

Assignment Interface for Lecturer (View Submission List)

Figure 6. Interfaces for lecturers

Assignment Interface for Student (Submit Assignment)

Assignment Interface for Student (Assess Assignment)



Example of Assessment Output for Student

Figure 7. Interfaces for students

V CONCLUSION AND FUTURE WORK

This paper has presented a tool for conducting a dynamic structural testing of a program executed in APA which integrates DyStruc-TDG, a test data generation method that have recently developed. This tool is expected to assist lecturers who have been teaching introductory programming courses in conducting a dynamic-structural testing on students' programming exercises. It also gives a systematic and consistent way of deriving test data among different individual lecturers regardless of having a particular knowledge of test cases design. In addition, this feature benefits the lecturers in terms of reducing their workloads, including the time spent for assessing programming exercises and averting them to construe the critical aspects of devising the suitable test cases to judge the correctness quality of students' programs. In this paper, it only highlights the design and implementation of the developed Auto-JPAT. As sub-sequence to a completion of the tool dvelopment, a users' experience evaluation was conducted from the contexts of users' perception and End-User Computing Satisfaction (EUCS).

As for the future work, there are some possible recommendations that can be realized especially in deriving the test cases design (test data). Currently, Auto-JPAT is only capable of performing dynamic-structural testing on programming solutions that involve only one specific class as its scope particularly for an introductory programming course. Generally, students of this course will be given with programming exercises that focus on one specific class only. However, it would be more beneficial if the tool could be extending by dealing with multiple classes. With this extended capability, the lecturers do not have to limit the possible solution that can be provided by the students.

Another possible recommendation is to include a negative testing criteria. DyStruc-TDG method that is integrated in Auto-JPAT does not include the negative testing while generating the test data. Negative testing refers to the testing of invalid data (input) against the program. This study does not concern of this issue as it is not critical in domain of APA since positive testing is sufficient to cover the required testing in APA. However, an ideal testing can be realized with the inclusion of the negative testing criteria.

ACKNOWLEDGEMENT

The authors acknowledge Ministry Higher Education FRGS Fund (Code SO: 12821) of Universiti Utara Malaysia for supporting this work.

REFERENCES

Al Shamsi, F., & Elnagar, A. (2012). An intelligent assessment tool for students' Java submissions in introductory programming courses, 4(1), pp. 59-69.

Auffarth, B., López-Sánchez, M., Campos i Miralles, J., & Puig, A. (2008). System for automated assistance in correction of programming exercises (SAC). In *International Congress University Teaching and Innovation (CIDUI)*, Lleida, Spain, pp. 104-113.

Blumenstein, M., Green, S., Nguyen, A. and Muthukkumarasamy, V. (2004). GAME: A Generic Automated Marking Environment for Programming Assessment, Proceedings of the International Conference on Information Technology: Coding and Computing (ITCC'04), Las Vegas, Nevada, Vol. 1, pp. 212-216..

Douce, C., Livingstone, D., & Orwell, J. (2005). Automatic test-based assessment of programming: A review. *Journal on Educational Resources in Computing (JERIC)*, *5*(3), 4.

Dow, S. (2011). How prototyping practices affect design results. *Interactions*, *18*(3), 54-59.

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

Ihantola, P. (2006). Creating and Visualizing Test Data from Programming Exercises. *Informatics in education*, *6*(1), 81-102.

Ihantola, P., Ahoniemi, T., and Karavirta, V. (2010). Review of Recent Systems for Automatic Assessment of Programming Assignments, *Proceedings of the 10th Koli Calling International Conference on Computing Education Research*, Koli, Finland, pp. 86-93.

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

Kitaya, H. & Inoue, U. (2016). An online automated scoring system for Java programming assignments. *International Journal of Information and Education Technology*, *6*(4), 275.

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

Latiu, G. I., Cret, O. A. & Vacariu, L. (2012). Automatic test data generation for software path testing using evolutionary algorithms. *Proceedings of Third International Conference on Emerging Intelligent Data and Web Technologies*, pp. 1-8.

Leung, H. K. & White, L. (1990). A study of integration testing and software regression at the integration level. *Proceedings of a Conference on Software Maintenance*, pp. 290-301.

Liang, Y., Liu, Q., Xu, J. & Wang, D. (2009). The recent development of automated programming assessment. *Proceedings of International Conference on Computational intelligence and software engineering*, pp. 1-5.

Lim, K. S., Lim, J. S., & Heinrichs, J. H. (2008). Validating an End-User Computing Satisfaction Instrument: a confirmatory factor analysis approach using international data. *Journal of International Technology and Information Management*, *17*(2), 6.

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

Nunome, A., Hirata, H., Fukuzawa, M. & Shibayama, K. (2010). Development of an e-learning back-end system for code assessment in elementary programming practice. *Proceedings of the 38th annual ACM SIGUCCS fall conference: navigation and discovery*, pp. 181-186.

Queirós, R. A. P. & Leal, J. P. (2012). PETCHA: a programming exercises teaching assistant. *Proceedings of the 17th ACM annual conference on Innovation and technology in computer science education*, pp. 192-197.

Rahman, K. A. & Nordin, M. J. (2007). A Review on the Static Analysis Approach in the Automated Programming Assessment Systems, *Proceedings of National Conference on Programming*, Kuala Lumpur, Malaysia, pp. 10-21.

Reid, S. (2005). The Art of Software Testing, Glenford J. Myers. Revised and updated by Tom Badgett and Todd M. Thomas, with Corey Sandler. John Wiley and Sons, New Jersey, USA, 2004. ISBN: 0-471- 46912- 2, pp 234. *Software Testing, Verification and Reliability*, *15*(2), 136-137.

Romli, R., Sulaiman, S., & Zamli, K. Z. (2015). Improving the reliability and validity of test data adequacy in programming assessments. *Jurnal Teknologi*, 77(9), 149–163.

Romli, R., Sulaiman, S., & Zamli, K. Z. (2014). Test data generation framework for Automatic Programming Assessment. *Proceedings of 8th Malaysian Software Engineering Conference*, pp. 84-89.

Romli, R., Sulaiman, S., & Zamli, K. Z. (2010). Automatic programming assessment and test data generation a review on its approaches. In *International Symposium in Information Technology (ITSim), 2010*, Vol. 3, pp. 1186-1192.

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

Sherman, M., Bassil, S., Lipman, D., Tuck, N., & Martin, F. (2013). Impact of auto-grading on an introductory computing course. *Journal of Computing Sciences in Colleges*, *28*(6), 69-75.

Tahbildar, H., & Kalita, B. (2011). Automated software test data generation: direction of research. *International Journal of Computer Science and Engineering Survey*, *2*(1), 99-120.

Tillmann, N., De Halleux, J., Xie, T., Gulwani, S., & Bishop, J. (2013). Teaching and Learning Programming and Software Engineering via Interactive Gaming. *Proceedings of the 2013 International Conference on Software Engineering*, pp. 1117-1126.