A Review on the Static Analysis Approach in the Automated Programming Assessment Systems

Khirulnizam Abd Rahman
khirulnizam@kuis.edu.my
Fakulti Teknologi & Sains Informasi
Kolej Universiti Islam Antarabangsa Selangor
Bandar Seri Putera, Bangi
43000 Kajang, Selangor

Md Jan Nordin
jan@ftsm.ukm.my
Fakulti Teknologi & Sains Maklumat
Universiti Kebangsaan Malaysia
43600 Bangi, Selangor

Abstract: *Automated programming assessment system (APAS) is a computer aided approach in checking and grading students programming exercises, without the hassle of doing it manually. There are two common approaches in assessing programming exercise automatically; dynamic and static. Dynamic approaches are done by executing the program source-code and the execution is test with a set of test data. Meanwhile, the static approaches are done by examining the source-code without execution. Discussion will cover advantages and disadvantages of using APAS and approaches implemented in most of the APASes. The review will be focusing on the methods of analysis that are categorized in the static analysis approach; such as programming style, program error (syntax or semantic), software metrics assessment, structural and non-structural similarity analysis, keywords analysis, plagiarisme detection and diagram assessment.*

Keywords: automated programming assessment, pseudo-code generator, pseudo-code comparison, static analysis, non-structural similarity.

1.0 INTRODUCTION

Automated programming assessment system (APAS) is gaining popularity in the field of teaching and learning programming (Ala-Mutka, 2005). The system is used to assess, evaluate, grade and manage students' programming exercises. It saves programming instructors from assessing the exercises manually.

There are many studies in this area since 1960 where Hollingsworth (1960) suggested an APAS to assess and evaluate assembly language programming exercises written on the punch card. Since then a lot of other APASes are developed to support different types of programming languages. This paper will be discussing the implementation of several techniques categorized in the static analysis approach of APASes. There are several techniques identified and discussed in the section 4.0.

*In this text APAS is referred as singular automated programming assessment system, while APASes is the plural.

2.0 ADVANTAGES OF USING APAS

There are many advantages of implementing APAS. The most significance advantage is saving lecturers time and workload. Manual programming exercise checking is tedious, repetitive work, mundane and time consuming. Lecturers need to conduct lectures, tutorials and lab sessions, prepare lecture notes, quizzes, exercises and check students assignments, consultation, project supervision, manage faculty or departments projects, do researches, writing articles, present research findings, attending seminars and conferences, and etc. These activities constraint them from going through each and every exercises submitted by their students.

Students need instant feedback (Venables & Haywood, 2001) to improve. After they submitted the exercises, there should be errors, incorrect programming style, or even misconception. They need to know mistakes they made, so they would not repeat them again. Since lecturers have a lot of works to do, they could not immediately check and grade the assignments. Such situation will not improve students' performance, and the lecturers could not monitor their students' performance. The system could handle the checking and evaluation very fast. Certain available system capable of giving immediate feedback on the students' assignments, only in few seconds, for example the ELP by Truong et. al. (2005) and *submit* by Venables and Haywood (2001).

In order to train the students to get used to the programming syntax, one way is by giving the students a lot of exercises. These intensive sets of exercises could also help them to solve programming problem. However it's hard to manage all the students' solution if the assignment is submitted manually. Sometimes the students submit via e-mail, or by using floppy disk, or even printed on papers. These kinds of mediums are difficult to manage. Thus, using the APAS could help you tremendously (Ala-Mutka, 2005). It will save you a lot of papers, prevent you from misplacing the students' answers and it will not make your room messy.

This system could also help to overcome bias (Norshuhaini et. al., 2006). Lecturers/instructors are human beings. They tend to be bias to his/her preferred students by giving more marks. The students that they don't like normally get less mark even though they performed very well. By using this system, mark will be given directly by the system without the interference of lecturers.

There are also some APASes that could do the plagiarism detection. Plagiarism detector such as the MOSS (Aiken, 2007), YAP (Wise, 1993) and JPlag (Prechelt et. al., 2000) could detect plagiarism even tough they change the variable name in the program code. This capability can help the lecturers to detect the plagiarism activity in the class. At least the lecturers can consult the students to avoid such activities again.

Although there are many advantages of implementing APAS, still many institution of higher learning implementing the manual style (Carter, 2003). This situation is due to the lack of user friendliness and hard to install. Most of the APASes do not support multiple programming languages. It is very important for an APAS to support multiple programming languages so that the assessment can be done centralized. For example ELP (Throung et. al., 2005) and BOSS (Joy & Luck, 1998) are solely supporting Java and Style++ (Ala-Mutka et. al., 2004) can only support C++. However the developers of APASes are doing their best to provide multiple programming languages. We can see this effort in the development of Ceilidh (Foxley et. al., 1996), CourseMaster (Symeonidis, 1998), WAGS (Norshuhaini et. al., 2006a) and few other APASes.

3.0 APPROACHES IMPLEMENTED IN AUTOMATED PROGRAMMING ASSESSMENT

For what ever reason APAS being developed, there are two major approaches implemented in the system. The approaches are dynamic and static analysis (Ala-Mutka, 2005). Most of the APASes available implement both approaches to complement each other (Rohaida, 2003). There are certain aspects the dynamic analysis is doing great jobs, and the rest are left to the static analysis to minimize processing burden.

3.1 Dynamic Analysis

Dynamic analysis approach involves code execution. The code execution will generate output and compared to the control output provided by instructor. In order to determine the code correctness and efficiency, the execution will also be tested against a set of test data (Foxley et. al., 1996).

Since this approach requires execution, processing overload might happen to the system's server. If uncontrollable, it can cause server breakdown and downtime. A prevention measure should be imposed to make sure the programs submitted by the student do not contain infinite loop or critical runtime error. This is to prevent system hang or crash. The programs need to be filtered from any viral elements such as Trojan horse, denial of service attack, or a backdoor script that could jeopardize the system security. All these security measures could result in complicated system development, thus more cost will occur.

This dynamic analysis approach has been discussed thoroughly by Rohaida (2003). In her thesis, she had classified dynamic analysis into three categories; black box testing, white box testing and regression.

3.2 Static Analysis

Programming assessment using the static analysis approach is a method to gather information about a programming code without having to execute it (Ala-Mutka, 2005). Basically the evaluation is done directly to the programming code.

There are a lot of techniques categorized in this approach such as programming style assessment, syntax and semantic error detection, software metrics analysis, structural similarity analysis, non-structural similarity analysis, keyword detector, plagiarisme detection, and diagram analysis. This approach could also uncover infinite loop, logical errors, and unused statements (Foxley et. al., 1996).

4.0 METHODS CATEGORIZED IN STATIC ANALYSIS

4.1 Programming Style Analysis

There are needs to check the students' programming writing style in order to promote programs' readability and maintainability. Among the criteria of highly readable program are indentation, structured, comments (for description), and meaningful variable names. These elements are very important for other people to easily understand the program and upgrade them in the future (Marini et. al., 2004).   That's why it is important to consider programs' readability to make sure it is a high quality program. Style++ is one of the systems developed to cater programming writing style quality for C++ language (Ala-Mutka et. al., 2004). It concentrates on the following criteria; comments (description), line spacing, variables naming, variable scope, program layout and the usage of constants.

4.2 **Error Detection (Syntax or Semantic)**

Syntax errors happen when the codes are written against the language grammar specification. Some common errors for the novice programmers are forgetting the semicolons in C or Java, if or for statements without the close braces ("}") and the use of wrong built in functions. Such errors could easily detected by most modern IDE (Integrated Development Environment) for the respective language. The modern IDEs are capable of detecting the error position and also giving suggestions for the corrections.

There are also special application such as CAP (*Code Analyzer for Pascal*) has been developed by Schorsch (1995) to detect syntax, logical and programming-style errors for Pascal programming language. CAP is very special because it capable of detecting the error and the error position in a very detail and precise manner. It also proposes a very comprehensive correction for the error found. It is a very helpful application to guide the students in learning Pascal and it acts as a tutor.

Semantic errors happen when the statements written are grammatically correct but logically wrong. Some common errors are the never-ending loops and division by zero. These errors will not produce error messages in the compiling process, however during the execution the application will produce something that we do not desire. Ceilidh (Foxley et. al., 1996) introduced semantic error detection that capable of detecting a suspicious never-ending loop statically. Since the Ceilidh is designed to evaluate programming exercise dynamically, it is very important to avoid never-ending loop. If such a program is allowed to execute in the system, it will consume a lot of the system memory and processing resources in order to execute a program that actually does not stop. It may also break the whole system.

4.3 **Software Metric Analyzer**

Software metrics is the measurement of some properties inside a software or program (Pressman, 2000). It uses numerical ratings to measure the complexity and reliability of source code, the length and quality of the development process and the performance of the application when completed (PCMag.com, 2007).

There are a lot of APASes developed to cater this type of analysis, Ceilidh, BOSS (Joy and Luck, 1998), Style++ (Ala-Mutka et. al., 2004), CAP (Schorsch, 1995) are among of them. For example, there is a software metrics called cyclomatic complexity suggested by McCabe (1976) is analyzed in many APASes such as in Assyst (Jackson dan Usher, 1997), ELP (Truong et. al., 2004) and Verilog Logiscope WinViewer (Mengel dan Yerramilli, 1999).

Rohaida and Mawarny (2006) also suggested a system to measure the complexity of Java programming exercises. This system focuses on the object oriented programming approach. The software metrics selected for the measurement are listed in Table 1.

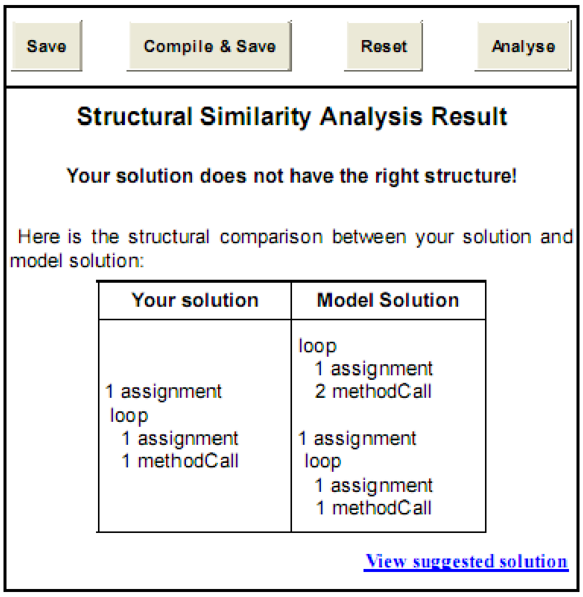
Software metrics	usage	Introduced by
Cyclomatic Complexity	Measures the sum of logical decision in the programs' module.	McCabe (1976)
NOC (Number of Classes)	Measures the number of class in a program.	Tegarden et. al. (1995).
SIZE2 (Number of properties)	Measures the number of properties in a program.	Li and Henry (1993)
AC (Attributes Complexity)	Calculate the value of attributes in a class based on a specific determined weight.	Chen and Lu (1993)
OpCom (Operation Complexity of Classes)	Calculate the value of complexity in the class behavior.	Chen and Lu (1993)

Table 1: The software metrics criteria measured in the system suggested by Rohaida and Mawarny (2006).

4.4 **Structural Similarity Analysis**

This is a technique to determine the similarity between the structures of students' program and the program solution provided by the instructor. It is done by generating the structure schema from both the students' answer and the instructors' solution, before they are compared.

In the research to develop Environment for Learning Programming (ELP), Thruong et. al. (2005) are using structural similarity analysis as a part of the static analysis assessment to evaluate students' programming answers. The structural similarity analisis is done by converting the program source code into the pseudo-code abstract. The pseudo-code abstract is the representation of the basic algorithmic structure of the program. The student's abstract representations then compared to the abstract representation generated from the instructor's solution models (the answer schemes). Since the abstract is too simple, the method proposed is only suitable to assess simple programming exercise. More complicated programming exercise will produce variation in students' answers. The instructors need to provide several different answer schemes in order to provide all the possible answers.



Structural Similarity Analysis Result

Your solution does not have the right structure!

Here is the structural comparison between your solution and model solution:

Your solution	Model Solution

1 assignment
loop
  1 assignment
  1 methodCall
	loop
	  1 assignment
	  2 methodCall
	
	1 assignment
	loop
	  1 assignment
	  1 methodCall

Figure 1: AST comparison.

The students' and the instructors' programs are translated into the AST (abstract syntax tree), and then marked up with the XML tags as suggested by Badross (2003). The AST then compared to decide whether they are match or not. Figure 1 is a sample of the generated AST.

According to the developers of ELP, the comparison process will produce only yes (for matching structure) or no (for un-matching structure). This is the main weakness of the system. The result is too objective. There should be a percentage value to indicate how much similar the student's program structure as compared to the instructor's program scheme structure.

4.5 Non-structural Similarity Analysis

The system is capable of comparing programming source code submitted by the students with the answer schemes provided by the instructor. The instructors need to provide more than one answer scheme in order to facilitate all the possible answer variation by the students. The students answer will be compared to all of the answer schemes provided. The highest mark from the comparison will be the mark for the students answer.

However, there is one weakness that can be observed from the comparison process in WAGS. In writing program, the variables may be declared anywhere in the program, as long as it is declared before used. It is better to separate all the variable declarations from the main program. Compare the students' variable declarations with variable declarations from the answer scheme. If it is done this way, we believe the comparison result will be better.

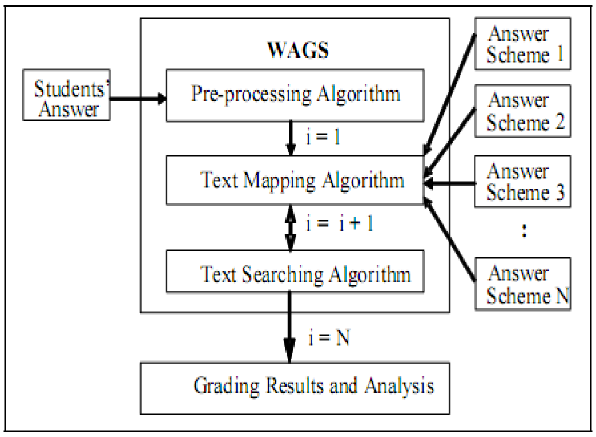


Figure 2: Solution proposed by Norshuhani et. al. (2006).

Khirulnizam et. al. (2007) suggested a very similar APAS to compare the similarity between student's answers and instructor's answer scheme. In order to produce a better similarity percentage result, both the programming answers from the student and also the lecturer are converted into the pseudo-code. After that, the variable declaration statements are separated from the pseudo-codes before they are compared. This is done because the variable declaration might be scattered in the program. The separation process could increase the similarity percentage. Refer to the Appendix A for the design of the system suggested.

Both structural and non-structural similarity analysis require the programming instructor to provide the programming answer schemes. The more of answer scheme variants provided, the more accurate the comparison will be. It is easy to provide all the answer schemes to the fundamentals programming questions, since the variant of the students answer will be quite limited. If the problems are bigger and more difficult, there will be many variant of solutions and answer from the students. It is impossible to provide the entire possible answer scheme for complicated problems. So both techniques are only suitable for small problems and just for the fundamental part of programming scope (Thruong et. al., 2004).

4.6 Keyword Analysis

This kind of assessment is done by searching the keywords used in the students' answers. Scheme-robo is one of the APASes that capable of searching the specific keywords used in the programming answer for problems written in one of a functional language, the Scheme (Saikkonen, 2001). Certain exercises need to be solved using a set of specific functions only determined by the instructor, while certain exercises are prohibited to use a certain functions in order to promote more creative solutions from the students.

4.7 Plagiarism Detection

Plagiarism is an unacknowledged copying of documents or programs (Joy & Luck, 1999). In programming exercise context a student is copying another program (from friends, books or internet sources) but assuming that the work is his/hers. Such activities does happen for whatever reasons, and it is a very time consuming task to check each and every single students' exercise in order to detect plagiarism.

Luckily there are several projects to detect plagiarism through automated checking. *Plague* (Whale, 1986) is among the early system developed to detect plagiarism in Pascal, Prolog, Bourne Shell and Llama. It uses a comparing algorithm (from the variant of Longest Common Subsequence - LCS) that compares the tokens from two sources of programming codes. However Plague did not survive long, because it is very complicated to modify the application to cater different programming (Granville 2002).

YAP (*Yet Another Plague*) is another system developed to detect plagiarism (Wise, 1992). In the latest version, YAP3 uses Wise developed a text matching algorithm known as RKR-GST the short form for *Running-Karp-Rabin Greedy-String-Tiling* (Wise, 1993). This algorithm is claimed to produce faster and better text matching result.

The same algorithm in YAP3 was improved and implemented in JPlag (Prechelt et. al., 2000). JPlag was developed using Java and by the time of this writing, it supports a list of programming languages such as C, C++, Java, Scheme and etc. The detail documentations, samples and free demonstration are available from http://www.ipd.ira.uka.de/jplag/.

MOSS (Measure Of Software Similarity) is another plagiarism detector (Aiken, 2007) for C, C++, Java, C#, Python, Visual Basic, Javascript, FORTRAN, ML, Haskell, Lisp, Scheme, Pascal, Modula2, Ada, Perl, TCL, Matlab, VHDL, Verilog, Spice, MIPS assembly, a8086 assembly, a8086 assembly, MIPS assembly and HCL2. It can be found at http://theory.stanford.edu/~aiken/moss/ .

4.8 Diagram Analysis

There are also special diagram analyzer that capable of checking the flowchart, object oriented design and simulating electronic circuits. Such features are available in CourseMaster (Symeonidis, 1998). The flowcharts will be evaluated by converting the flowchart submitted by the students into a BASIC codes. The BASIC codes than sent to the dynamic tools and executed for the dynamic evaluation. For the object oriented design evaluator, the student's design will be tested for completeness, correctness, accuracy, the use of correct relationships between the various components, the use of classes that are needed to complete the design, and etc. And for the logical circuit simulator, students' logical circuit is sent to the CircuitSim to be evaluated through a set of circuit simulation processes.

5.0 ADVANTAGES AND DISADVANTAGES OF IMPLEMENTING STATIC ANALYSIS APPROACH

There are a lot of advantages is implementing static approach in APAS. The most significant advantage is it is much cheaper and easier to develop and deploy (Truong et. al., 2004). Static analysis does not require complicated algorithm or code to develop compared to dynamic analysis, thus the development cost could be reduced. Since it does not require code compilation and execution, it will surely reduce the server workload (Truong et. al., 2004).

Dynamic analysis could only be done by compiling and executing the code, so only error-free program can be submitted by the students. However static analysis approach could do the analysis against the programming answer even though the code has errors (Rohaida, 2003). Since there's no requirement to execute the code, there's no need to generate testing input data (Rohaida, 2003). It also gives the flexibility to the students' answers (Norshuhaini et. al., 2006), where it helps to promote students' creativity.

There are also few disadvantages of static analysis approach to be considered. Lectures/instructors also need to provide more solution schemes in order to make the similarity comparison works effectively (Truong et. al., 2004). Without the multiple solution schemes, the system is not capable to handle the variety of solutions by the students.

The biggest drawback of static analysis approach is that we could not evaluate the correctness and the efficiency of the programming solution submitted by the student (Ala-Mutka, 2005). This is very important criteria in program evaluation. Yet, it could only been solved through dynamic analysis approach.

6.0 CONCLUSION

There are two major approaches in automated programming assessment: dynamic and static. Categorized in the static analysis approach are programming style assessment, syntax and semantic error detection, software metrics analysis, structural similarity analysis, non-structural similarity analysis, keyword detector, plagiarisme detection, and diagram analysis. However there is a major drawback in the static approach: the incapability to determine program correctness and functionality. There should be more research on this matter.

7.0 REFERENCES

Aiken, A. 2007. *A System for Detecting Software Plagiarism* [cited August 2007]. Available from http://theory.stanford.edu/~aiken/moss/.

Ala-Mutka, K, Uimonen T., Järvinen H. M. 2004. Supporting Students in C++ Programming Courses with Automatic Program Style Assessment. *Journal of Information Technology Education,* Vol 3.

Ala-Mutka, Kirsti M. 2005. A Survey of Automated Assessment Approaches for Programming Assignments. *Computer Science Education* 15 (June 2005):83-102.

Badros, Greg. *JavaML : An XML-based Source Code Representation for Java Programs* 2000 [cited 06 Mei 2007]. Available from http://badros.com/greg/JavaML/.

Carter, J., English, J., Ala-Mutka, K., Dick, M., Fone, W., Fuller, U., & Sheard, J. 2003. How shall we assess this? ACM SIGCSE Bulletin, 35(4), 107 – 123.

Chen, J. Y. and Lu, J. F. 1993. A New Metric for Object-oriented Design. *Information Software Technology* Vol 35 (April 1993):232–240.

Chu, H. D., Dobson , J. E. and Liu I.C.. 2006. *FAST-A Framework for Automating Statistic-based Testing* . 1997 [cited Jun 2006]. Available from http://citeseer.nj.com/73306.html.

Foxley, E. Higgins, C, & Gibbon, C. 1996. *The Ceilidh System : A General Overview.* [cited August 207] . Available from http://www.cs.nott.ac.uk/CourseMarker/more_info/html/Overview96.htm

Granville, Andrew. 2002. Detecting Plagiarism in Java Code, University of Sheffield. Available from http://www.dcs.shef.ac.uk/intranet/teaching/projects/archive/ug2002/pdf/u9arg.pdf

Halstead, Maurice H. 1977. *Elements of Software Science*. North-Holland: Elsevier.

Hollingsworth, J. 1960. Automatic Graders for Programming Classes. *Communication of the ACM* 3 (10):528-529.

Jackson, D. & Usher, M. 1997. Grading Student Programs Using ASSYST, Proceedings of the 28th ACM SIGCSE Technology Symposium on Computer Science Education.

Joy, M & Luck, M. 1998. *The BOSS System for On-line Submission and Assessment* [cited 2007]. Available from http://www.ulster.ac.uk/cticomp/joy.html

Joy, M and Luck, M.. 1999. Plagiarism in Programming Assignments. *IEEE Transactions On Education* 42 (2, May 1999):129.

Khirulnizam Abd Rahman, Syarbaini Ahmad, Md Jan Nordin. 2007. The Design of an Automated C Programming Assessment Using Pseudo-code Comparison Technique. Paper read at National Conference on Software Engineering and Computer Systems, at University Malaysia Pahang.

Marini Abu Bakar, Norleyza Jailani, Sofian Idris. 2002. *Pengaturcaraan C*. Kuala Lumpur: Prentice Hall.

McCabe, T. J. 1976. A Complexity Measure. *IEEE Transaction of Software Engineering* 4 (SE-2):308-320.

Norshuhani Zamin, Emy Elyanee Mustapha, Savita K.Sugathan, Mazlina Mehat, Ellia, and Anuar. 2006. Development Of A Web-Based Automated Grading System For Programming Assignments Using Static Analysis Approach. Paper read at International Conference on Electrical and Informatics, at Bandung, Indonesia.

PCMag.com. *Software Metrics*. 2007 [cited August 2007]. Available from http://www.pcmag.com/encyclopedia_term/0,2542,t=software+metrics&i=51690,00.asp.

Prechelt L, Malpohl G., Philippsen M. 2000. Finding plagiarisms among a set of programs with JPlag. *Journal of Universal Computer Science*.

Pressman, R. S. 2000. *Software Engineering: A Practitioner's Approach*: McGraw-Hill.

Saikkonen, R., Malmi , L., and Korhonen, A.. 2001. Fully Automatic Assessment of Programming Exercises. Paper read at ITiCSE2001.

Rohaida Romli. 2003. Penjanaan Data Ujian untuk Penaksiran Automatik Tugasan Aturcara C, Universiti Kebangsaan Malaysia.

Rohaida Romli , Mawarny Rejab 2006. Penyukatan Automatik Kekompleksan Tugasan Aturcara Java. Paper read at National ICT Conference at Universiti Teknologi MARA, Arau, Perlis, Malaysia.

Rohaida Romli, Mazni Omar, Cik Fazilah Hibadullah. 2004. Automatic Correctness Assessment Program for Java Programming Assignment. Paper read at M^{2}USIC 2004, at Multimedia University, Malaysia.

Schorsch, Tom. 1995. CAP: An Automated Self-Assessment Tool To Check Pascal Programs For Syntax, Logic And Style Errors. Paper read at SIGCSE ’95 at Nashville.

Shafer, S. C. 2005. LUDWIG: An Online Programming Tutoring and Assessment System. *inroads – The SIGCSE Bulletin* 37 (June 2005):56-60.

Symeonidis, P. 1998. An in-depth Review of CourseMaster’s Marking Subsystem.

Tegarden, D. P. , Sheetz, S. D. , Monarchi, D. E.. 1995. A Software Complexity Model of Object-oriented Systems. *Decision Support System* Vol 13:241-262.

Truong, N., Roe, P., Bancroft, P.. 2003. A Web Based Environment for Learning to Program. Paper read at Proceedings of the 26th Australasian computer science conference.

Truong, N., Roe, P., Bancroft, P.. 2004. Static Analysis of Students’ Java Programs. Paper read at 6th Australian Computing Education Conference (ACE2004), at Dunedin, New Zealand.

Truong, N., Roe, P., Bancroft, P.. 2005. Automated Feedback for "Fill in the Gap" Programming Exercises. Paper read at Australasian Computing Education Conference, at Newcastle, Australia.

Venables A., Haywood L. 2003. Programming students NEED instant feedback! Paper read at Conferences in Research and Practice in Information Technology.

Wei Li, Sallie Henry 1993. Object-oriented Metrics that Predict Maintainability. *Journal of Systems and Software* 23:111-122.

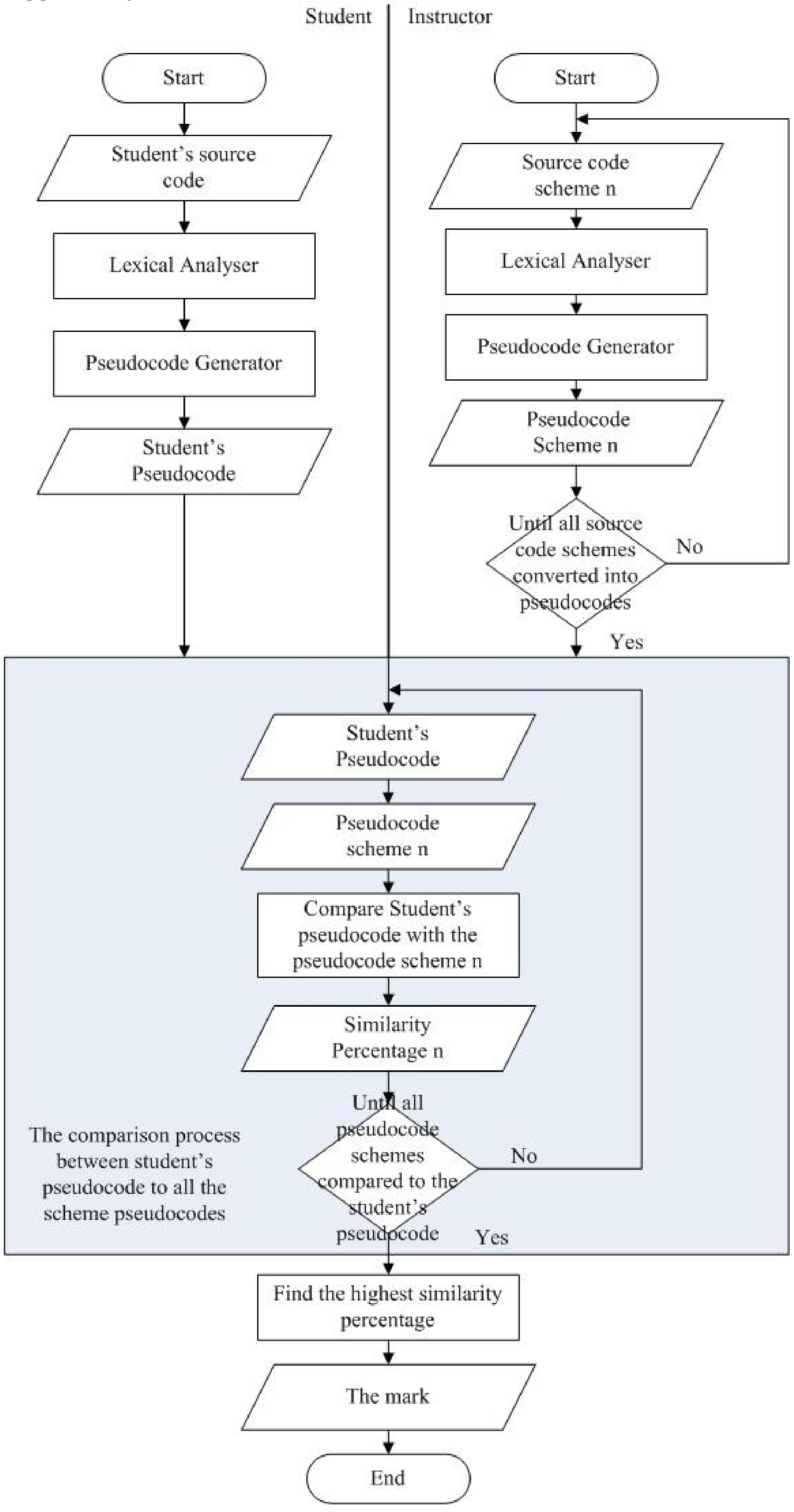
Whale, Geoff. 1986. Detection of Plagiarism in Student Programs. Paper read at 9th Australian Computer Science Conference, at Canberra.

Wise, M. J. 1992. Detection of Similarities in Student Programs: YAP'ing maybe preferable to Plague'ing. ACM SIGCSE Bulletin, 24, March 1992:268-271.

Wise, M. J. 1993. String Similarity via Greedy String Tiling and Running-Karp-Rabin Matching. [cited June 2007]. Available from ftp://ftp.cs.su.oz.au/michaelw/doc/RKR_GST.ps .

Yerramilli, Susan A. Mengel and Vinay. 1999. A Case Study Of The Static Analysis Of The Quality Of Novice Student Programs Paper read at SIGCSE '99

APPENDIX A : The design of the automated assessment using the pseudo-code comparison technique suggested by Khirulnizam et. al. (2007).


Student
Instructor
Start
Student's source code
Lexical Analyser
Pseudocode Generator
Student's Pseudocode
Start
Source code scheme n
Lexical Analyser
Pseudocode Generator
Pseudocode Scheme n
Until all source code schemes converted into pseudocodes
No
Yes
Student's Pseudocode
Pseudocode scheme n
Compare Student's pseudocode with the pseudocode scheme n
Similarity Percentage n
Until all pseudocode schemes compared to the student's pseudocode
No
Yes
The comparison process between student's pseudocode to all the scheme pseudocodes
Find the highest similarity percentage
The mark
End