Modern programming assignment verification, testing and plagiarism detection approaches

Aleksejs Grocevs
Department of Software Engineering
Riga Technical University
Riga, Latvia
e-mail: root@smallserver.org

Natālija Prokofjeva
Department of Software Engineering
Riga Technical University
Riga, Latvia
e-mail: natalija.prokofjeva@rtu.lv

***Abstract*—This paper provides insights of possible plagiarism detection approach based on modern technologies – programming assignment versioning, auto-testing and abstract syntax tree comparison to estimate code similarities.**

***Keywords*—automation; assignment; testing; continuous integration**

INTRODUCTION

In the emerging world of information technologies, a growing number of students is choosing this specialization for their education. Therefore, the number of homework and laboratory research assignments that should be tested is also growing. The majority of these tasks is based on the necessity to implement some algorithm as a small program. This article discusses the possible solutions to the problem of automated testing of programming laboratory research assignments. The course “Algorithmization and Programming of Solutions” is offered to all the first-year students of The Faculty of Computer Science and Information Technology (~500 students) in Riga Technical University and it provides the students the basics of the algorithmization of computing processes and the technology of program design using Java programming language (the given course and the University will be considered as an example of the implementation of the automated testing). During the course eight laboratory research assignments are planned, where the student has to develop an algorithm, create a program and submit it to the education portal of the University. The VBA test program was designed as one of the solutions, the requirements for each laboratory assignment were determined and the special tests have been created. At some point, however, the VBA offered options were no longer able to meet the requirements, therefore the activities on identifying the requirements for the automation of the whole cycle of programming work reception, testing and evaluation have begun.

I. Plagiarism detection approaches

To identify possible plagiarism detection techniques, it is imperative to define scoring or detecting threshold. Surely it is not an easy task, since only identical works can be considered as “true” plagiarism. In all other cases a person must make his decision whether two pieces of code are identical by their means or not. However, it is possible to outline some widespread approaches of assessment comparison.

A. Manual Work Comparison

In this case, all works must be compared one-by-one. Surely, this approach will lead to progressively increasing error rate due to human memory and cognitive function limitations. Large student group homework assessment verification can take long time, which is another contributing factor to error-rate increase.

B. Diff-tool Application

It is possible to compare two code fragments using semi-automated *diff* tool which provides information about Levenshtein distance between fragments. Although several visualization tools exist, it is quite easy to fool algorithm to believe that a code has multiple different elements in it, but all of them are actually another name for variables/functions/etc. without any additional contribution.

C. Abstract Syntax Tree (AST) comparison

Abstract syntax tree is a tree representation of the abstract syntactic structure of source code written in a programming language. Each node of the tree denotes a construct occurring in the source code. Example of AST is shown on Fig. 1.

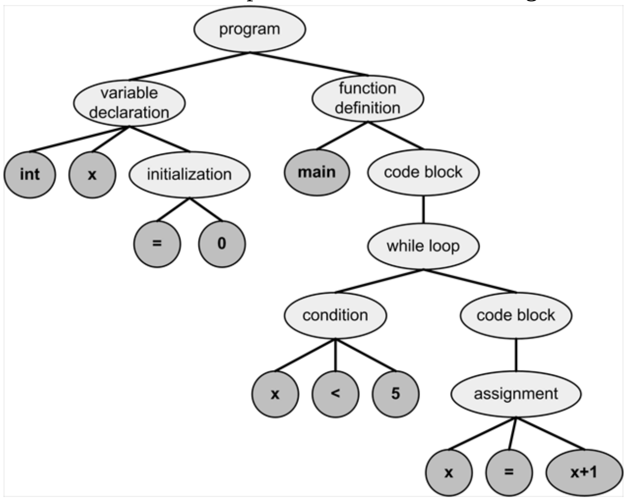


Fig. 1. Code parsing into abstract syntax tree

Since the code comparison technology takes very important part in the work of plagiarism detection and software evaluation, some authors assume that software plagiarism mainly appears as copy-and-paste or with a little modification after this, which will not change the function of the code, such as replacing the name of methods or variables, reordering the sequence of the statements etc. [2], [3].

To generalize AST creation approach, multiple ready-to-use libraries exist, most notable are ANTLR (a powerful parser generator for reading, processing, executing, or translating structured text or binary files), JavaCC (much more a Parser generator than a compiler. AST support is provided through another lib called JJTree) and SableCC (parser generator which generates fully featured object-oriented frameworks for building compilers, interpreters and other text parsers. In particular, generated frameworks include intuitive strictly-typed abstract syntax trees and tree walkers).

It is possible to implement any of abovementioned AST generators, however ANTL provides wider source code language support and it will be used for our reference system implementation.

II. Verification system metric identification

In order to get a grade, the student has to implement the required algorithm as a program, submit the program in a binary (compiled) form as well as providing its source code. The professor has to test both the program using a subset of pre-calculated input/output data pairs, and its source code equivalence to the binary representation, including the compilation ability and the absence of errors while executing it. It is imperative to evaluate the factors affecting the whole process and to choose the metrics for intercomparison [1].

A. *Metrics and Identification Process*

Algorithm implementation validation speed is criteria that reflects how fast can the professor obtain the program and its source code, execute and test it using the predefined test patterns. In the University, the student has to upload the result of his work to the "Homework" section of the Moodle education portal, where the professor should run and evaluate it after downloading.

Plagiarism check – even if the tasks are relatively similar it is crucial to make sure the source code was written by the student himself and not plagiarized from a colleague. This check should be made once by a professor in order to avoid the code consecutive altering so it can successfully pass the check.

Evaluation quality – even in case of simple tasks, such as "implementing a list sorting algorithm" there might be many border cases which can lead to incorrect execution results, but sometimes may not be checked due to time limitation. When using the automated testing solutions, it is possible to pre-create the large number of different tests that check all the aspects of the performance of the given assessment.

Report preparation – it is important to put the data of successful/unsuccessful test runs together and to inform the student of the result. When using on-line solutions, there is a possibility to send a test-passed-successfully notification as soon as an assessment has been uploaded and tested. It is also important to record the summary of all students' assessments in form of a table that can later be used to be uploaded to the University education portal.

Safety check – the binary representation of the program and its source code are usually tested separately in order to save time, and this is most commonly done in that particular order. Although, it is not always possible to quickly detect the malicious code, which is meant to erase or alter the test results or even the testing system itself, when the solution consists of several files or modules. Therefore, while testing the compiled programs they should be treated as potential malware or viruses that may damage the test environment. Ideally, they should be executed in the sandbox environment which ensures the isolation of the potential threat.

Bug fix tracking – if the code has been partially altered (either on the professor's request or during the debugging process) the modified parts of the file are indistinguishable from the previous code in the file. Therefore, the professor has to manually compare two versions of the file in order to detect these changes, or even look through the entire source code file. This problem can be solved by using version control system (VCS). The Moodle system itself does not provide neither an option nor plugin for that, so this possibility can only be considered in the context of implementing it in on-site solution.

B. *Available automated verification systems*

At the moment, all of the assessment checking, test executing and running as well as plagiarism check/percentage evaluation is done manually. It is obvious that an ordinary human cannot keep the source code of five hundred similar programs in mind, nor is he able to measure their similarity. That is why similar researches exist that also suggest automated testing implementation [7].

In order to aid the manual testing the academic department of the University has developed a VBA script that automatically executes and checks the assessments for a compiled program; the test results are recorded in an Excel file. This improvement has allowed to speed up the evaluation process making it possible for the professor to focus more on the source code, which is a more important than an actual test run.

Some authors [4], [5] are offering to use the automated verification of test runs and a plagiarism check as a separate Moodle module, which implements many previously defined criteria at the time the assessment is uploaded to Moodle. However, none of these researches provide the complete solution to this problem.

In addition to the above-mentioned methods it is also possible to use the features of the Continuous Integration technology that was made specifically to constantly check the software. This would allow the students to upload their works into an automated testing environment that would immediately check these assessments based on the tests prepared by a professor and would provide the near-realtime response, whether the program has passed the tests or not. In addition to the functional check this service could also evaluate the source code for potentially harmful behavior or actions in an isolated sandbox-environment.

III. Anti-plagiarism integration possibilities

By identifying the possible solutions and considering the current approach to the testing it is possible to compare these approaches using the previously proposed metrics in accordance to further anti-plagiarism system integration.

A. Manual Testing

The speed and the quality of manual check of each assessment will be definitely lower compared with the automated check. The numerical representation of absolute comparison results is impossible in that case due to average human concentration and performance abilities of each individual.

Report preparation is done manually, therefore both the error rate and the time consumed will be considerably higher when compared to an automated check.

Plagiarism check in general will be less accurate with a large amount of assessments, however human perception makes detecting such cases reflexively or by using additional environment information possible (the plagiarism rate is higher if the students are friends). Furthermore, biased perception and evaluation of the work are also possible due to the human factor.

Safety check depends on the set of rules enforced by the University. These rules define how to verify the programming assessments and how to run the executable files. It is most likely that a professor will primarily check the compiled program in order to return it for correction in case of a program failing the tests. This saves him the time for compiling the program from its source code.

B. Custom Script Usage

The speed and the quality of such checks are increased in comparison with manual checks. Since the VBA implementation allows to check just one assessment at a time, it cannot be considered as a finished testing automation, as the required programming work has yet to be copied to the proper folder for a script check.

Report preparation is consistent with the expectations and the requirements – the results of automated test are recorded in Excel file for each student.

The plagiarism and safety checks remain at the same level as in case of manual check, since the VBA script can only work with the compiled program.

C. E-Learning Portal Approach

According to the authors, the speed and the quality of such checks are high and take place in real time.

The preparation of report is made using Moodle so this solution is also consistent with the requirements.

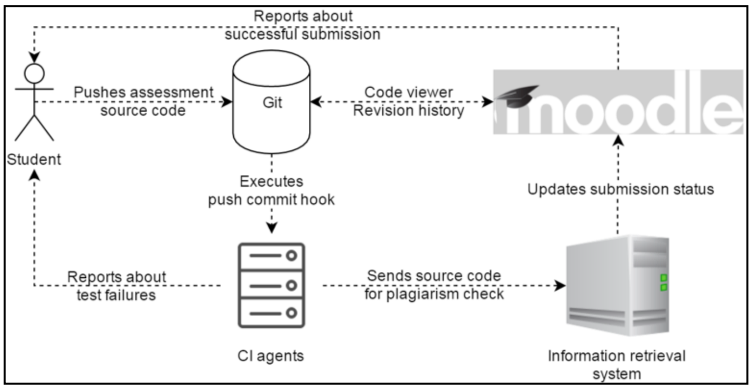
The plagiarism check is present in some announced solutions but it is not fully functional (system improvement work is still in progress).

Safety check remains poor since the sandbox-environment is not used, which increases the risk of compromising the whole Moodle system (since all tests are executed under the Moodle user privileges).

D. Custom Implementation

While creating an on-site solution using the third-party developer tools, it is possible to meet all the abovementioned needs. We propose to use Gogs as a Git-repository – the storage for the source code of all programming assessments; Jenkins as a Continuous Integration server; Docker as a sandbox-environment and Apache Solr or other full-text search system as a plagiarism checker. Many authors [6], [7], [8] have approached the problem of detecting the plagiarism in the source code, and some authors [9] are proposing the solutions that are consistent with the University requirements and can be integrated into the suggested infrastructure. Our suggested infrastructure involves the workflow shown on Fig. 2:

Fig. 2. Overall verification workflow schema


Reports about successful submission
Student
Pushes assessment source code
Git
Code viewer
Revision history
moodle
Executes push commit hook
Updates submission status
Reports about test failures
Sends source code for plagiarism check
CI agents
Information retrieval system

Workflow step description:

- The students code the task in the program code and uploads/updates it using Git;
- Jenkins checks all the students' repositories once per minute, sends a plagiarism check request to Sorl whenever a new commit appears, creates a separate sandbox-environment, compiles a program within it and runs the tests. If the tests are passed – Moodle API is used to mark the student's assessment as successfully completed and to upload the source code his behalf. If the tests are not passed or the harmful code was found – the student is notified by an e-mail and has to go back to step one. The Docker sandbox container is being automatically removed either when the tests are completed or by timeout.
- After the assessment submission deadline, the professor sets all Git-repositories to read-only mode, runs a Jenkins-plugin, which sends a plagiarism check request for each of the assessments to Solr and records the additional data on plagiarism score for every task's source code to Moodle.
- The only action left for the professor is to check the source codes or to compare the latest source code version with the previous using the built-in Gogs tools.

Such an approach provides the level of checking speed and quality as well as a report preparation level similar to that of the Moodle plugin.

It is possible to use outer systems for plagiarism checks since Jenkins API allows to connect the various external services.

Safety check is at a high level due to the isolated container (sandbox) use, so the risk of system infestation by the malicious code and other destructive actions is minimized.

Tracking of error correction is a standard feature of the Gogs-repository which facilitates the comparison of file versions and change-tracking.

IV. Conclusion and further work

Based on the comparison results of the programming work automated testing capabilities using the VBA script, Moodle plugins and on-site solution, we can assert that the initial implementations in that area already exist; many authors are highlighting this fact. However, there is no general method which could improve the education quality by simplifying the verification process and shifting the assessor's professional skill focus from routine tasks towards the student's skills check in implementing the required task and understanding of the material. Our suggested approach to the programming assessment storage organization, testing and verification allows to solve the abovementioned problems and improve the professor's working efficiency.

Hereafter we are planning to implement our suggested AST-based plagiarism checking approach alongside with other mentioned techniques in a single system.

References

[1] Grocevs A., Prokofjeva N, “The Capabilities of Automated Functional Testing of Programming Assignments,” In: Procedia – Social and Behavioral Sciences, vol. 228, pp. 457-461, 2016.

[2] Cui, B., Li, J., Guo, T., Wang, J., Ma, D., “Code Comparison System based on Abstract Syntax Tree,” 2010 3rd IEEE International Conference on Broadband Network and Multimedia Technology (IC-BNMT), Beijing, pp. 668-673, 2010.

[3] Hansen, D. M., “Paperless subjective programming assignment assessment: a first step,” Journal of Computing Sciences in Colleges, vol. 29(1), pp. 116-122, 2013.

[4] Cheng, Z., Monahan, R., Mooney, A., “nExaminer: A semi-automated computer programming assignment assessment framework for Moodle,” International Conference on Engaging Pedagogy (ICEP11) NCI, Dublin, Ireland, 2011.

[5] Thiébaut, D., “Automatic evaluation of computer programs using Moodle's virtual programming lab (VPL) plug-in,” Journal of Computing Sciences in Colleges, vol. 30(6), pp. 145-151, 2015.

[6] Cosma, G., Joy, M., “An approach to source-code plagiarism detection and investigation using latent semantic analysis,” Computers, IEEE Transactions on, vol. 61(3), pp. 379-394, 2012.

[7] Pozenel, M., Furst, L., Mahnicc, V., “Introduction of the automated assessment of homework assignments in a university-level programming course. In Information and Communication Technology, Electronics and Microelectronics (MIPRO), 2015 38th International Convention on, IEEE, pp. 761-766, 2015.

[8] Zhang, D., Joy, M., Cosma, G., Boyatt, R., Sinclair, J., Yau, J., “Source-code plagiarism in universities: a comparative study of student perspectives in China and the UK. Assessment and Evaluation in Higher Education, vol. 39(6), pp. 743-758, 2014.

[9] Kikuchi, H., Goto, T., Wakatsuki, M., Nishino, T., “A Source Code Plagiarism Detecting Method Using Sequence Alignment with Abstract Syntax Tree Elements,” International Journal of Software Innovation (IJSI), vol. 3(3), 41-56, 2015.