AUTOMATED SUBMISSION CHECKING: IMPROVING REMOTE LEARNING ECOSYSTEM FOR PROGRAMMING CLASSES

Agnes Leung[1], Maxim Mozgovoy[1], Evgeny Pyshkin[1]

[1]*University of Aizu (JAPAN)*

Abstract

The project contributes to the quality of software engineering education by producing a practical submissions assessment-supporting system, aimed at college and university teachers. Checking code conformance to the specification and detecting plagiarism (unauthorized “borrowings” of code fragments) consumes much time and energy of the teachers, and lenient grading procedures make a negative impact on students’ discipline and lowers motivation of course participants. While plagiarism detection and software testing are well-known tasks, there are few efforts to develop an open-source submission assessment system that would fit online course management system-supported workflows. Our objective is to address this shortcoming by designing and deploying a robust teacher-centric submission assessment system suitable for practical use in programming classes, seamlessly integrated with the learning management system, and based on the state-of-the-art source code analysis algorithms and code similarity visualization mechanisms facilitating quick evaluation.

Keywords: distant learning, programming, automated assessment.

1 INTRODUCTION

The digitally transformed world requires transformation of teaching and learning practices. Particularly, programming and software development-related classes require many specific activities besides traditional lectures and exercises, hence, assuming higher degrees of interactivity and collaboration compared to individual assignments [1] (as shown in Figure 1). Such activities are typically supported by a variety of tools including learning management systems, online meeting tools, version control, and submission assessment systems, the latter two being often integrated with each other. Assessment of student contributions may require plagiarism detection tools, which are particularly helpful in the case of limited direct communication. Academic contributions to distant learning need to be understood from the perspective of daily teacher and student needs, so that to enable the workflow making the whole ecosystem of software engineering instruction better.

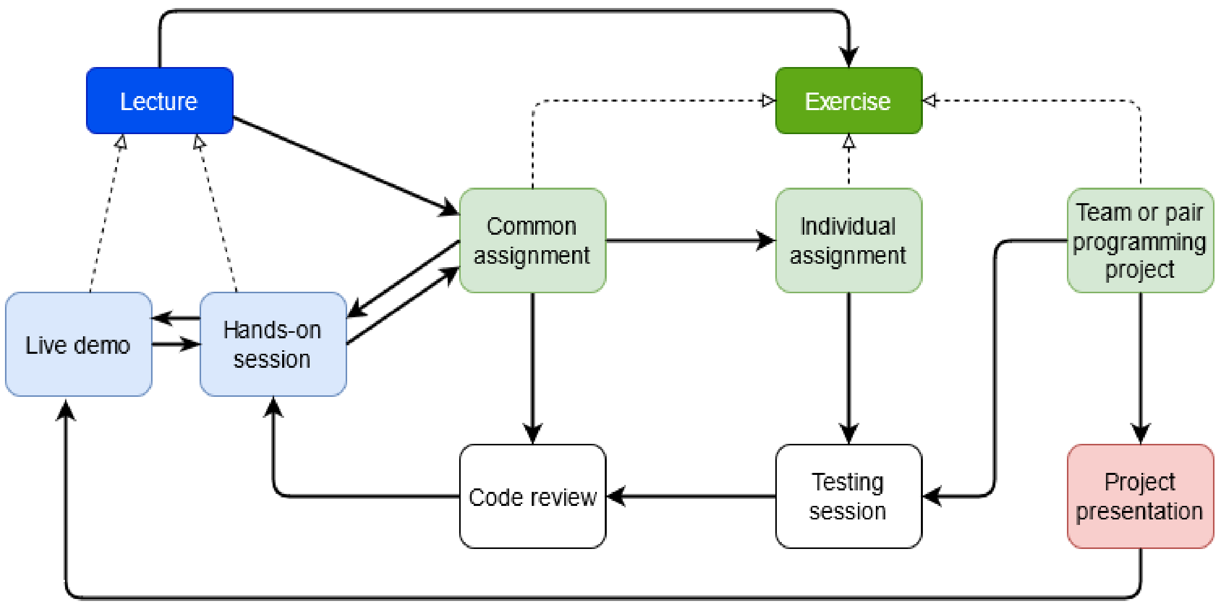

Figure 1. Variety of programming class activities (adopted from [1]).

The use of online education supporting systems is now wider than ever due to the ongoing pandemic. The majority of higher educational institutions rely on instruments such as Zoom and Moodle to support learning activities of remote students. While these systems provide necessary tools for basic needs of teachers and learners, many daily scenarios still require much manual work. In the case of programming courses, the most prominent of them are compilation & testing and plagiarism checking of student-submitted assignments. Individual tasks comprising this routine are regularly addressed in research literature. However, surprisingly little effort is made to integrate state-of-the-art methods into a working system, suitable for practical use in computer technology-supported courses. Figure 2 presents the major use cases, which need to be supported by an automated submission assessment system.

Most existing systems that perform automated assessment of student-supplied computer programs (and usually limited to checking their conformance to the problem specification) are “online judges”, designed to support programming competitions [2]. Thus, they cannot be directly applied in a conventional classroom due to differences in design goals. Plagiarism detections systems, such as Turnitin [3], are typically targeted at detecting overlapping online sources for a given document such as an essay or research paper. This task is different from source code plagiarism detection, since the students typically copy their code from their peers and predecessor rather than search from online sources, because it is hard to find an exact solution to the specific assignment unless it is a well-known classic algorithm [4]. Some recently developed instruments are specifically tailored for detecting duplications in offline collections of software code [5-6], but most related research is focused on purely algorithmic aspects of the problem, such as robust file comparison procedure, high speed of detection process or explicit support for specific programming languages [7-8]. Visualization [9] and reporting capabilities, as well as integration with the automated grading instruments [10], crucial for efficient educational use, are often neglected. As a result, teachers still usually rely on manual detection, which is a very time-consuming and laborious process. Furthermore, source code is easy to modify, refactor or obfuscate, which makes detection more challenging.

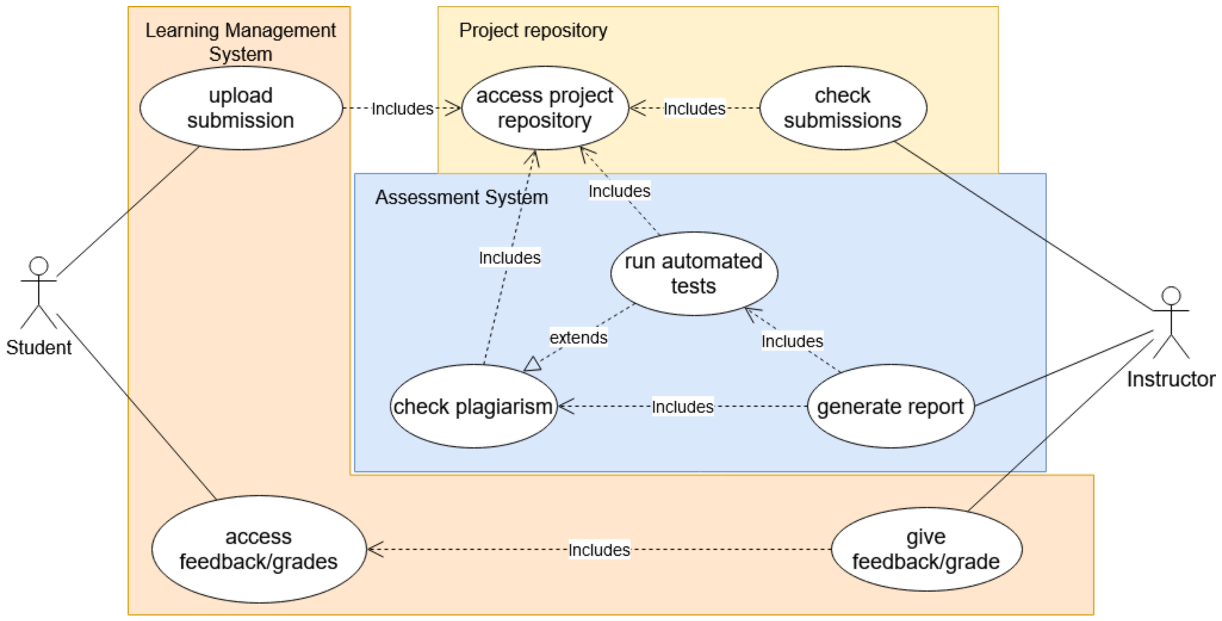

Figure 2. Submission assessment workflow: major use cases.

2 PROJECT SCOPE AND METHODOLOGY

Introducing the tools for checking software code for illegal “borrowing” is not trivial both from methodological and technical perspectives. It is commonly presumed that the students must work on their projects independently and avoid collaboration. Such an approach favors developers’ individual creativity but does not help in training soft skills and abilities to collaborate. Next, engineering education aims to advance students’ abilities to understand and re-use successful practices and standard solutions. This assumption may sometimes be against pedagogical goals. Finally, there are few open-source submission assessment and plagiarism detection systems that are designed specifically for online courses. Though there are significant achievements in code similarity checking algorithms, their classroom adoption is not straightforward, their compatibility with major online submission systems and automated code testing instruments is not guaranteed. We also need convenient visual interfaces helping the teachers to interpret the source code analysis results easier.

2.1 Research Questions and Project Goals

In the course of the project, we seek to answer the following questions

1) How the teachers of online programming courses actually check student submissions, and how an automated assessment system can be seamlessly integrated into this process?
2) What kind of reporting is the most convenient/useful for the teachers, and provides the most efficient support in the grading process?
3) How such a system can be designed, what are the relevant state-of-the-art algorithms, and how to integrate our solution into existing educational scenarios without much effort?

The practical goal of the proposed project is to automate teachers' daily routine tasks that consume much time and energy. We believe that the existence of a convenient submission assessment instrument can be a noticeable contributing factor for better quality of courses offered at educational institutions, especially where large groups of students are involved. The teachers will have more time to focus on course content, and the students will have a clear understanding of the assessment criteria and better chances for fair grading.

2.2 System Architecture

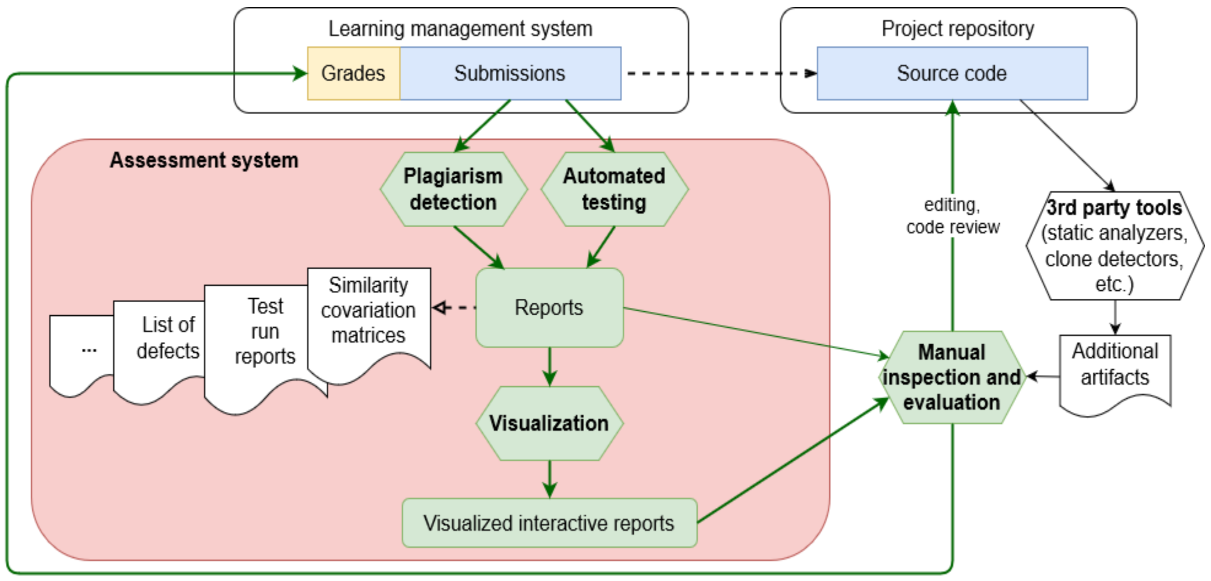

Figure 3. Major elements of automated submission assessment system.

As shown in Figure 3, the assessment system takes submissions from the learning management system (e.g., Moodle), checks if the source code works, and whether it overlaps with any other student's source code. Then the system generates interactive reports as a reference for teachers to check students' submissions. The assessment system consists of three main components, namely plagiarism detection, automated testing, and visualization.

2.2.1 Plagiarism Detection

The principles of source code plagiarism depend on the type of exercise.

In Figure 1, programming exercises can be categorized into three major types, including common assignments, individual assignments and team programming projects. Common assignments are the exercises, where all students get the same task. However, they also require students to work independently and develop their independence as programmers. This also implies that we expect that each student will submit their original work. Individual assignments are similar to common assignments, but the tasks are personalized. Lastly, team programming projects aim to train students' soft skills as a developer. In addition, according to the assignment's difficulty, teachers might provide templates [4] for students to follow and complete the solution.

The plagiarism principles of common assignments might differ from that of the other two types of exercises. For checking common assignments, we would like to see whether the students borrow code from other students, including the participants enrolled in the previous year programming courses. On the other hand, students get different tasks in individual assignments and team programming projects. Checking whether students have solutions similar to other students in the same class might be ineffective. However, the tasks are not guaranteed to be different from the previous year exercises.

Due to the above reasons, a submission assessment system should provide a number of options that would facilitate possible adjustments of plagiarism detection principles to the practical requirements teachers inspecting and evaluating the student source code submissions. Examples of such options could be the existence of templates and used programming languages. Specifically, in [4], the following necessary features are suggested:

- template highlighting;
- source code comparison against the submissions of classmates (so that to discover possible clusters of borrowed source code);
- source code comparison to the submission from the past courses;
- support for different programming languages;
- code tokenizing, for achieving higher detection accuracy.

2.2.2 Integration with Automated Testing Tools

To verify the correctness and quality of the solutions submitted by students, the automated testing and verification tools are naturally used in a programming class. Integration of submission assessment system with automated testing tools may significantly enhance the process of fair evaluation of student submissions. Such an integration is not only helpful for checking whether the code fits the requirements. The testing results (such as similar failed tests) may be one of factors to detect the similarity between different submissions.

2.2.3 Visualization

The above two components, plagiarism detection and automated testing are meant to assist teachers by automating repetitive tasks and provide results, but surely not to replace teachers. Therefore, teachers should still be responsible to inspect the results and provide feedback on the submissions. As the components generate machine-friendly reports, it is necessary to visualize them for easier analysis.

3 ONGOING PROJECT AT A GLANCE

Our current primary focus is software code plagiarism detection in student assignments, submitted to a Moodle-based course management system. We are aiming at making as few presumptions about the course structure as possible, but some preliminary considerations are inevitable:

- We presume that the course is separated into “topics” (or “weeks”, or “chapters”), associated with separate programming assignments.
- We presume that each student has to submit a number of exercises for each topic. These exercises are individual and bound to a specific topic. In other words, we do not address group work and cross-topic “project” assignments that are supposed to undergo revisions and updates.
- We presume that the students use Moodle to submit their submissions as file attachments (via a built-in Assignment activity). This proposal seemingly goes against a widespread practice of keeping source code on repository hosting platforms like Github, but for the scenarios we consider (small individual weekly assignments), using the standard Moodle submission management tools is arguably a reasonable choice. Furthermore, it is not difficult to update the system to handle external download links rather than file attachments.
- We presume that each individual submission is stored in a separate zip file attachment.

Most course management systems implement certain instruments for 3rd-party component integration, for example, via Learning Tools Interoperability (LTI) technology [11]. However, using LTI might require changes in the current configuration of the course management system used, which is not always an option. Thus, we are compelled to rely on conventional external APIs, typically available in default configurations. For instance, Moodle provides a REST API service, which can be used to access student submissions.

Similarity or plagiarism detection is performed with a well-known open-source tool JPlag [12]. In addition to high detection quality, JPlag offers features such as optional tokenization (a technique used to overcome plagiarism obfuscation), support of different programming languages, support of code templates (pieces of code to be excluded from detection), and flexible reporting capabilities. In particular, JPlag can produce machine-readable CSV reports that can be used to build interactive visual reports that we are planning to provide to the user.

4 CONCLUSIONS

Since the practical goal of the project is to automate teachers' daily routine tasks, we believe that a convenient submission assessment instrument can be a noticeable contributing factor for better quality of the courses offered at educational institutions, especially where large groups of students are involved. The teachers will have more time to focus on course content, and the students will have better chances for fair grading. Our project mostly relies on existing methods reported in literature and available as open-source solutions. Various software similarity detection algorithms are well known, but they implement different approaches (fingerprinting, string matching, tree matching) and may provide different degrees of reliability in a classroom setting. Most existing systems cannot be directly integrated into online course management systems like Moodle. We need to evaluate suitability of known algorithms and integrate them into a conventional submission grading interface featuring a visual reporting tool for easier identification of potentially plagiarized submissions requiring more thorough analysis.

The project is still ongoing, and we anticipate a number of further practical tasks to work on. These tasks include:

a) Designing the system GUI aimed at facilitation the work of end users. User controls supporting downloading submitted files from the learning management system, storing credentials, checking plagiarism, etc. may be beneficial for getting more convenience for teachers.

b) Integration with existing plagiarism detection systems (such as JPlag).

c) Supporting stub functionality for automated code testing.

d) Analysis and visualization of the plagiarism detection system reports linked to grading subsystem.

ACKNOWLEDGEMENTS

This project is supported by the University of Aizu research funding (AY2020 competitive research funding, grants P-30, P-34).

REFERENCES

[1] E. Pyshkin, "On programming classes under constraints of distant learning," *The 4th International Conference on Software and e-Business (ICSEB-2020)*, Dec 18-20, 2020, Osaka, Japan. In press.

[2] S. Wasik, M. Antczak, J. Badura, A. Laskowski, and T. Sternal, "A survey on online judge systems and their applications," *ACM Computing Surveys (CSUR)*, vol. 51(1), pp. 1-34, 2018.

[3] M. Halgamuge, "The use and analysis of anti-plagiarism software: Turnitin tool for formative assessment and feedback", *Computer Applications in Engineering Education*, vol. 25(6), pp. 895-909, 2017.

[4] M. Mozgovoy and E. Pyshkin, "Plagiarism Detection Systems for Programming Assignments: Practical Considerations," *The 15th International Conference on Software Engineering Advances (ICSEA 2020)*, Oct 18-22, Porto, Portugal, IARIA, pp. 16-18, 2020.

[5] C. Ragkhitwetsagul, J. Krinke, and D. Clark, "A comparison of code similarity analysers," *Empirical Software Engineering*, vol. 23(4), pp. 2464-2519, 2018.

[6] D. Kılınç. et al, "Overview of Source Code Plagiarism in Programming Courses," *International Journal of Soft Computing and Engineering*, vol. 5(2), pp. 79-85, 2015.

[7] M. Mozgovoy, *Enhancing Computer-Aided Plagiarism Detection*, The University of Joensuu, 2007, 131 p.

[8] M. Mozgovoy, S. Karakovskiy, and V. Klyuev, “Fast and Reliable Plagiarism Detection System,” *Frontiers in Education'07*, Milwaukee, USA, 2007.

[9] M. Freire, “Visualizing program similarity in the Ac plagiarism detection system,” *Proceedings of the working conference on Advanced Visual Interfaces*, pp. 404-407, 2008.

[10] J. C. Rodríguez-del-Pino, *Virtual Programming Lab*, Retrieved from https://moodle.org/plugins/mod_vpl.

[11] IMS Global. *Learning Tools Interoperability*. Retrieved from http://www.imsglobal.org/activity/learning-tools-interoperability

[12] L. Prechel, G. Malpohl, and M. Philippsen, “Finding plagiarisms among a set of programs with JPlag,” *Journal of Universal Computer Science*, vol. 8(11), pp. 1016-1038, 2002.