IIAI Open Conference Publication Series
IIAI Letters on Informatics and Interdisciplinary Research
Vol.002, LIIR082
DOI: https://doi.org/10.52731/liir.v002.082

# Teacher-oriented Source Code Similarity Detection and Visualization for Programming Assignments

Maxim Mozgovoy [*], Evgeny Pyshkin [*], John Blake [*],
Marina Purgina [*], Agnes Leung [*]

## Abstract

In programming classes, instructors need to work with numerous exercise submissions to verify whether the submitted source code meets the requirements, and whether there is any unauthorized borrowing of code fragments. The checking procedure is laborious requiring much unproductive effort and time. However, ignoring instances of potential plagiarism may negatively impact learner motivation. Despite the existence of practical tools developed for software testing and similarity detection, there are still issues in developing an open-source submission assessment system that would streamline the classroom workflow. This paper describes a practical submission assessment system that reduces the time teachers spend checking the solutions submitted by students.

*Keywords:* programming instruction, source code similarity, classroom workflow automation, similarity visualization.

## 1 Introduction

Educational institutions responded to societal lock-downs in 2020-22 with a significant increase in the use of digital and online teaching and learning platforms. Naturally, programming and software development-related classes are no exception in this extensive transformation of teaching practices involving using learning management systems (LMS) such as Moodle, online meeting tools, testing frameworks, version control, and bug tracking systems. What makes this transformation challenging specifically for software education is that software-related classes require many activities which involve high degrees of interactivity and collaboration [1][2].

Even when checking students' submissions using submission management automation provided by Moodle, much time is still spent on source code testing and similarity detection. Some available plagiarism detection tools (such as JPlag [3], MOSS [4], or Plaggie [5]) are very helpful but still require class instructor to complete a lot of manual operations.

Figure 1 displays the major use cases of a source code-based submission assessment process, which helped us to identify the features needed for a practical system. Although

[*] University of Aizu, Aizu-Wakamatsu, Japan

there are tools supporting different submission checking activities, there are few attempts to incorporate these tools into a working teacher-oriented system to be used in a classroom.

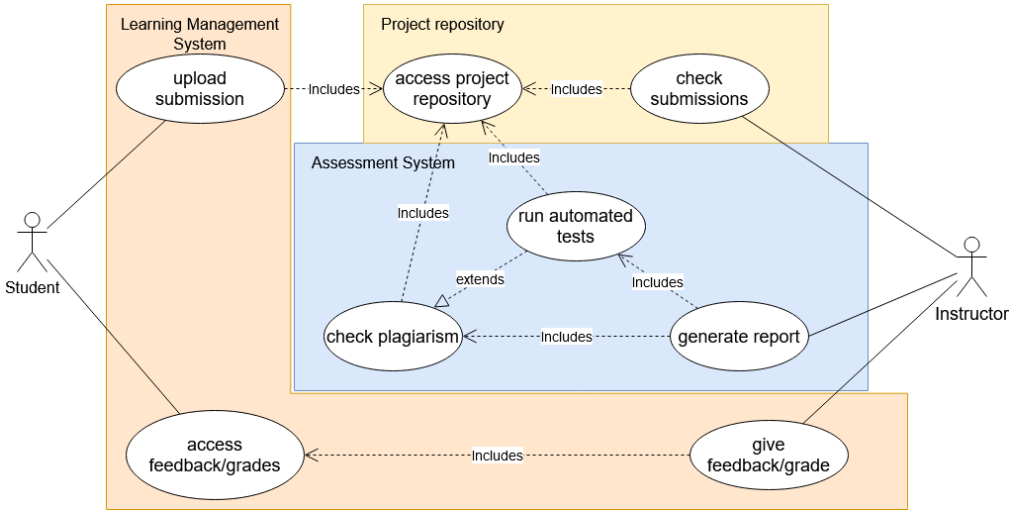


Figure 1: Major use cases in checking submissions (adapted from [1]).

## 2 Related Work

**Source code compilation and testing.** Automated assessment of computer programs is a common feature of online judges usually designed for use in programming competitions [6][7]. Such systems enable automated checking of the submitted programs against the requirement specification (normally, using the set of available test cases and under the limits of execution time constraints). Online judge systems play an important role in evaluating the correctness of software created by programming learners. However, they do not completely suit the needs and goals of programming classes, which are different from programming competitions.

**Plagiarism detection** Plagiarism detection tools are essentially similarity detection tools, since the decision of whether plagiarism has occurred is made by the person checking the similarity results. This decision depends on whether credit is given to the original author in the form of a citation. Conventional plagiarism detection tools such as Turnitin [8] are commonly aimed at detecting online duplicates for strings within a given essay or academic paper. The task of natural language plagiarism detection shares common aspects with source code software plagiarism elicitation, though the latter has serious particularities. For example, as it may be impossible for students to find the exact solution from online sources, students can still copy them from their peers instead [9]. Additionally, for source code plagiarism detection, we are interested in finding similarities on the structural and functional levels, rather than exact text matches. Even though approaches concerning finding similarities in mathematical equations on a structural level that were discussed in [10][11] provide certain insights in detecting similar source code fragments, they could not be directly applied to the source code structural similarity, since there are solutions based on the given patterns or object structures that should not be considered as instances of plagiarism. There are tools developed for detecting duplicates within a collection of student submissions, but they mainly target the algorithmic aspects of plagiarism detection.

What is lacking is a streamlined software similarity detection system that requires little effort on the part of the classroom instructor. Features such as providing visualization capabilities [12] and integration with LMS help reduce the workload of the teachers.

# 3 System at a Glance

Our current implementation is designed to retrieve submissions from the LMS (e.g. Moodle) and detect duplicates found in the submissions of other students. Although it would be ideal to have a system that supports as many course structures as possible, to keep the development work moving, a few presumptions are unavoidable for a pilot study [1]:

- The course has a clear division of topics (or weeks, chapters, etc.), where each topic has its own programming exercises.
- The exercises are submitted individually and are bound to a specific topic. Additionally, exercises that require revisions are not discussed.
- The exercises are submitted to Moodle as zip files.

In this paper, we describe two major components of the system: the plagiarism detection unit based on JPlag algorithms, as well as the interactive visual interface facilitating the teacher's work.

## 3.1 Plagiarism Detection

Depending on the learning objectives, the source code plagiarism principles may vary.

**For individual assignments,** one or more programming tasks are usually suggested to the students. Depending on the class size, students might be assigned to the same programming task, but are still expected to produce original independent solutions. Therefore, while checking the submissions, we are interested to detect the cases, when the students borrow parts of the source code from their peers in their class or previous cohorts.

**In team or pair programming projects,** the academic focus is complemented by training students' soft skills through collaboration with their classmates and solving the problems together. In such cases, detecting plagiarism might be less effective or even unnecessary, although we still might be interested in possible code similarities with submissions from previous cohorts.

**Scaffolded projects** provide a specific case, when the projects developed by students are based on some source code templates [13]. The content of the provided templates needs to be excluded from the plagiarism detection process.

With respect to the practical needs of programming classes, a plagiarism detection module should provide the following important capabilities [13]:

- Support for various programming languages;
- Source-code tokenization;
- Source-code analysis at a local level (i.e. comparison against other code submissions rather than against online resources); and
- Template highlighting.

### 3.2 Visualization

Plagiarism detection and testing components help in automating routine tasks; however, the goal of assessment system is not to replace the teachers. The instructors are still responsible for the detailed inspection of the students' source code and providing feedback on students' submissions. Interactive visual interface facilitates in-depth consideration of source code analysis results.

## 4 Grading Cat: Workflow and Similarity Visualization

*Grading Cat* is an open source software implementing the proposed system in Python and available at https://github.com/rg-software/grading-cat.

Currently, it supports the following key features:

1. Downloading submissions from Moodle (using Moodle REST API service);
2. Calculating source code similarities used for plagiarism detection; and
3. Visualizing submission similarities.

Figure 2 draws the workflow of the submission assessment process. The important stages are as follows:

1. Open or create a project from the main application window.
2. Manage the settings using the configuration editor.
3. Download the Moodle submissions to sync the new submissions with the data source.
4. Choose an assignment to launch plagiarism detection tool and calculate source code similarities.

Figure 3 shows the interface for visualizing the source code similarities. On a visual graph, each node corresponds to a submission, the brighter lines represent the detected similarity, which is higher than the set threshold. The threshold can be easily adjusted using the provided slider control.

In the process of similarity detection, detailed reports are generated. These reports can be accessed directly by clicking the nodes on the visual graph. Figure 4 shows an example of a detailed report containing information about found duplicated fragments, number of tokens covered, along with the highlighted source-code fragments facilitating the instructor's work and decision making.

Similarity calculation is implemented based on using plagiarism detection tool JPlag [3], which is a popular solution assuring high detection quality [14] and providing good features such as tokenization with an option to tune the sensitivity, work with code templates, as well as support for different programming languages (including Java, C#, C, C++ and Python3). In JPlag, to calculate the program similarity of a pair of programs, the programs are first converted into token strings, then JPlag uses the Greedy String Tiling algorithm [15] to check the percentage of token strings (i.e. program similarity) that can be covered.

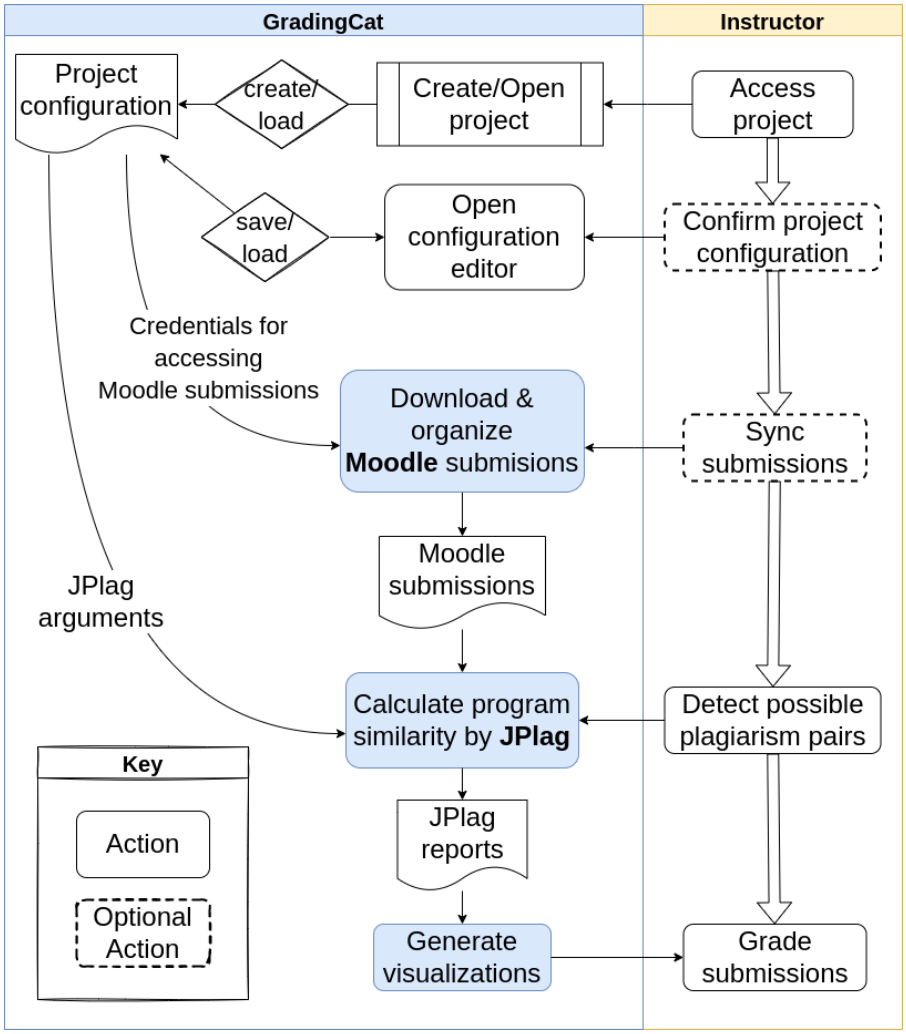


Figure 2: Submission assessment workflow with *Grading Cat*.

## 5 Conclusion

Introducing a convenient submission assessment tool into the classroom enhances the quality of programming courses, particularly in the case of large groups of students. The instructors could spend more time on improving course content, rather than on tedious work of extensive checking of all submitted solutions for potential plagiarism and failure to meet requirements. In turn, the students benefit from shorter grading time and increased chances for fair evaluation.

This work does not address the improvement of the core detection functionality of the existing plagiarism detection software. It focuses on their smooth adoption for the practical needs of programming classes with a particular emphasis on the creation of instructor-oriented interfaces. These interfaces aim at connecting existing tools for source code analysis to interactive visual components for similarity evaluation.

In its current version, the system still requires a number of extensions and features to be implemented for further integration with the automated tools for submitted source code compilation and testing. Automated testing does not only facilitate checking the confor-

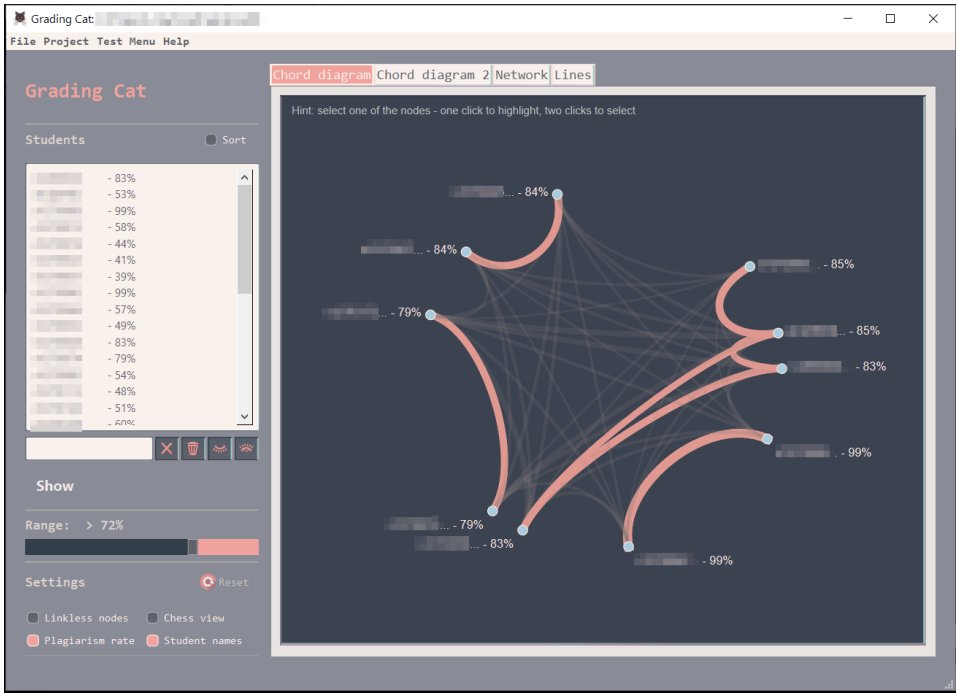


Figure 3: A screenshot of Grading Cat's visualization.

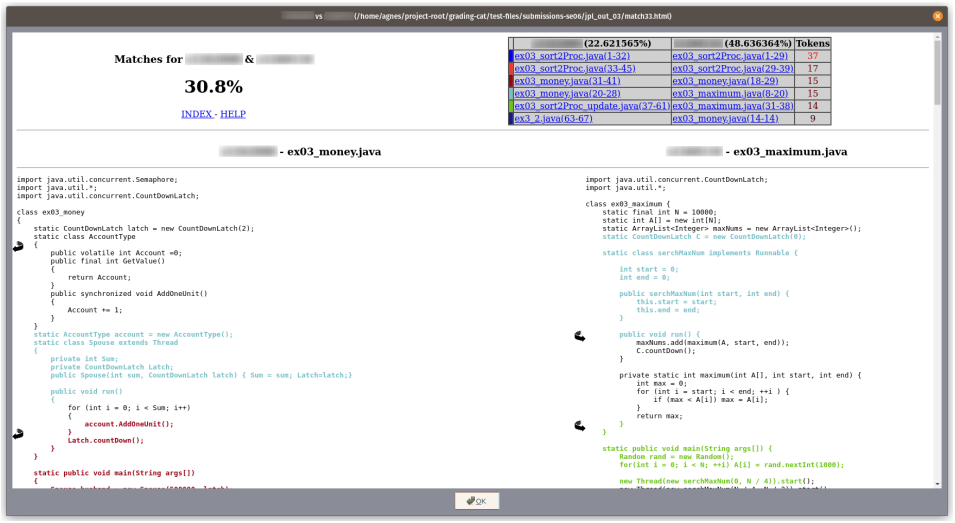


Figure 4: A screenshot of a detailed report.

mance of the submitted solutions to requirements, but enables further analysis of behavioral similarity, which, in turn, can contribute to the improvement of the source code similarity detection algorithms [16].

The integration of submission assessment systems with automated testing tools is an important pedagogical issue as well. On the one hand, such an integration follows the standard software quality assurance practices to be introduced in programming classes. On the other hand, using automated tests should enhance the fairness of the submission evaluation process while providing better feedback to learners; thus, in a sense, making the whole

teaching process effectively multimodal, an important issue in present-day learning systems and environments [17][18].

Though our pilot studies demonstrate good applicability of the system to the practical needs of programming classes, the current work still lacks an evaluation stage. We trust plagiarism detection quality of our tool, since it is based on well-established foundation, but the coverage and the adequacy of our current "user stories" have to be examined further.

## References

[1] A. Leung, M. Mozgovoy, and E. Pyshkin, "Automated Submission Checking: Improving Remote Learning Ecosystem For Programming Classes," Proc. INTED2021, IATED, 2021, pp. 4946–4951; doi:10.21125/inted.2021.1004.

[2] E. Pyshkin, "On Programming Classes under Constraints of Distant Learning," Proc. 2020 The 4th International Conference on Software and E-Business, 2020, pp. 14–19; doi:10.1145/3446569.3446574.

[3] L. Prechelt, G. Malpohl, and M. Philippsen, "Finding plagiarisms among a set of programs with JPlag," J. Univers. Comput. Sci., vol. 8, no. 11, 2002, p. 1016.

[4] A. Aiken, "Moss – A System for Detecting Software Similarity," Feb. 2023; https://theory.stanford.edu/ aiken/moss/.

[5] A. Ahtiainen, S. Surakka, and M. Rahikainen, "Plaggie: GNU-licensed source code plagiarism detection engine for Java exercises," Proc. 6th Baltic Sea Conference on Computing Education Research: Koli Calling 2006, ser. Baltic Sea '06. New York, NY, USA: Association for Computing Machinery, 2006, pp. 141–142; doi:10.1145/1315803.1315831.

[6] S. Wasik, M. Antczak, J. Badura, A. Laskowski, and T. Sternal, "A Survey on Online Judge Systems and Their Applications," ACM Computing Surveys, vol. 51, no. 1, 2018, pp. 3:1–3:34; doi:10.1145/3143560.

[7] Y. Watanobe, M. M. Rahman, T. Matsumoto, U. K. Rage, and P. Ravikumar, "Online judge system: Requirements, architecture, and experiences," Int. J. Soft. Eng. Knowl. Eng., vol. 32, no. 06, 2022, pp. 917–946; doi:10.1142/S0218194022500346.

[8] M. N. Halgamuge, "The use and analysis of anti-plagiarism software: Turnitin tool for formative assessment and feedback," Computer Applications in Engineering Education, vol. 25, no. 6, 2017, pp. 895–909; doi:10.1002/cae.21842.

[9] M. Mozgovoy, "Enhancing Computer-Aided Plagiarism Detection," Ph.D. Thesis, Joensuun yliopisto, 2007.

[10] K. Yokoi and A. Aizawa, "An approach to similarity search for mathematical expressions using MathML," Proc. Towards a Digital Mathematics Library, Grand Bend, Ontario, Canada, July 8-9th, 2009, pp. 27–35.

[11] E. Pyshkin and M. Ponomarev, "Mathematical equation structural syntactical similarity patterns: A tree overlapping algorithm and its evaluation," Informatica, vol. 40, no. 4, 2016, pp. 377-385.

[12] M. Freire, "Visualizing program similarity in the Ac plagiarism detection system," in Proceedings of the Working Conference on Advanced Visual Interfaces, ser. AVI '08. New York, NY, USA: Association for Computing Machinery, 2008, pp. 404–407; doi:10.1145/1385569.1385644.

[13] M. Mozgovoy and E. Pyshkin, "Plagiarism Detection Systems for Programming Assignments: Practical Considerations," Proc. ICSEA 2020, IARIA, 2020, pp. 16-20.

[14] M. Novak, M. Joy, and D. Kermek, "Source-code similarity detection and detection tools used in academia: a systematic review," ACM Transactions on Computing Education (TOCE), vol. 19, no. 3, 2019, pp. 1–37; doi:10.1145/3313290.

[15] M. J. Wise, "String similarity via greedy string tiling and running Karp-Rabin matching," Online Preprint, Jan 1993, vol. 119, no. 1, 1993, pp. 1–17; https://www.researchgate.net/publication/262763983.

[16] H. Cheers, Y. Lin, and S. P. Smith, "Academic source code plagiarism detection by measuring program behavioral similarity," IEEE Access, vol. 9, 2021, pp. 50391–50412; doi:10.1109/ACCESS.2021.3069367.

[17] C. Jewitt, "Multimodality and literacy in school classrooms," Review of research in education, vol. 32, no. 1, 2008, pp. 241–267; doi:10.3102/0091732X07310586.

[18] M. Dressman, "Multimodality and language learning," The handbook of informal language learning, Wiley, 2019, pp. 39–55; doi:10.1002/9781119472384.ch3.