Plagiarism Detection Systems for Programming Assignments: Practical Considerations

Maxim Mozgovoy and Evgeny Pyshkin
University of Aizu

Tsuruga, Ikki-Machi, Aizu-Wakamatsu, Fukushima, 965-8580, Japan
Email: {mozgovoy,pyshe}@u-aizu.ac.jp

***Abstract*—We discuss a project contributing to the quality of software engineering education by producing a state of the art code duplication and plagiarism detection system, aimed at college and university teachers. Though detecting plagiarism (as unauthorized "borrowings" of code fragments) consumes time and energy of the teachers, ignoring this issue makes a negative impact on students' discipline and lowers motivation of course participants. While plagiarism detection in software code is a well-known research task, there are no open source modern plagiarism detection systems that are designed for actual classroom use by implementing state of the art detection techniques, convenient visualizations, integration with course management systems, and supporting common use case scenarios.**

***Keywords*–Education; plagiarism; software; programming; exercises; online learning.**

I. Introduction

Societal lock-down of 2020 caused by Covid-2019 outspread pushed educational institutions to enforce teaching and learning processes based on active using of online and distant education technology. Distant learning tools have many advantages (flexibility in using them at any time from virtually any device, convenient access, possibilities for user collaboration, creating less stressful study process), but also considerable limitations. That's why methodological and organizational solutions for distant learning should be developed so that they would improve the teaching process in a way serving both remote and traditional face-to-face teacher/student communication. Further digitalization of learning process does not mean its transfer to online environments only, developing better computer-assisted tools to support traditional teaching is equally important.

Programming is commonly considered as an activity favoring outsourcing, task distribution and online communication, which make the project work of experienced engineers more efficient and well organized. However, in programming teaching, the lack of direct in-person communication often makes difficult to fairly check the solutions of students and to support them in their practical work. Even within the scope of traditional face-to-face classes, it is hard to manage all the students' projects, especially if you have big groups of students. Thus, instructors need good instruments to support practical exercise assignments and their checks.

Teachers who conduct online classes, as we observe, typically do not seek to use a single universal tool covering all their needs. Instead, they tend to choose the most suitable software for the given task. For example, a teacher might publish video recordings on YouTube, communicate with students via Slack, organize videoconferences using Zoom, and publish learning materials on Moodle. Some activities are, however, easier to organize than others. Videoconferencing and messaging is a relatively straightforward process with today's technology, while conducting exercise sessions and ensuring proper homework evaluation can be more laborious. Since the students may work on their assignments in an unsupervised environment (and organizing a proper proctoring procedure might be complicated and not always desirable), ensuring fair study conditions for everyone and reducing the amount of cheating is essential.

Plagiarism is a type of academic dishonesty that consists in reusing documents composed by other authors without proper acknowledgement of the original source [1]. Plagiarism is a common phenomenon among students, as some of them may tempt to reuse solutions developed by their peers or copied online, thus, eliminating their educational value.

Many authors note that plagiarism can often be prevented by designing better, more personalized assignments, training students to avoid unintentional plagiarism, and administrative measures [2], [3]. It is also often suggested that preventive measures should be combined with computer-assisted plagiarism detection practices. Most detection tools, such as Turnitin [4], are designed for detecting overlapping online sources for a given document (an essay or a research paper), but some recently developed instruments are specifically tailored for detecting duplications in offline collections of software code [5].

II. Plagiarism Detection in Student Projects: Challenges and Practical Goals

Plagiarism detection in student-submitted software source code is an established research topic, stemming from the task of identifying duplicated fragments in large software systems, which has its own importance, since the duplicated code impairs project architecture, conditions worse project maintainability, increases the risk of software bugs, and may seriously affect code efficiency. The problem of detecting software code improper reuse differs from natural language plagiarism detection due to several factors:

- Code collections under testing are typically available offline, since it is nearly impossible to find online a fragment of code that solves a particular teacher-supplied task, unless it is a well-known classical algorithm;
- Students typically receive the same or very similar assignments, so they tend to borrow code from their peers or predecessors;
- Software code is easy to modify, refactor or obfuscate, keeping its functionality intact;
- In principle, code reuse itself is a common engineering practice, which does not always refer to plagiarism.

Thus, though it seems obvious to apply some NLP algorithms suitable for general-purpose text processing, we have to mention a number of important issues demonstrating serious particularities of plagiarism detection in software programs.

Contradiction between standard software engineering practices and pedagogical aims. Teachers fight plagiarism, since, first, it itself contradicts major pedagogical aims. However, at the same time (specifically, in engineering disciplines), teachers should introduce existing standard solutions. Software engineering practices naturally favor reusing the successful solutions and standardized models, but the proper reuse should be differentiated from large scale source code copying.

Detecting structural source code similarity. In process of comparing the source code fragments, we are not so interested in finding exact matches, but their structural resemblance and/or functional equivalence. One may expect that recognizing structural similarity of math equations may provide useful insights in detecting structurally similar source code fragments [6], [7], the latter being nothing but the texts in a formal language. However, especially in the cases when students are encouraged to use structural software organization and presentation models (such as software design patterns, specific project architectures, etc.), we could not accuse a student in plagiarism, based on structural similarity only.

Few solutions may be used in classroom environments. Surprisingly, there are very few systems designed specifically for actual classroom use. Most researchers tend to be 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. Therefore, the tasks of designing appropriate user interfaces or providing specific functionality relevant for teachers of programming classes is challenging, and often neglected.

Most commercial solutions are for texts in natural languages. The gap between the achievements in developing algorithms for source code processing and their applicability to classroom environments is not filled by commercial companies offering solutions focused primarily on detecting matches for natural-language texts. As a result, teachers still often rely on manual detection, which is a very time-consuming and laborious process, they could not gain all the possible benefits from using the online submission systems (such as Moodle and/or automated code testing / grading instruments).

Thus, the problem of identifying reliable similarity detection methods, suitable for academic environment and applicable to a wide range of programming languages is far from being resolved. That is why, in sum, the practical goal of our project is to automate teachers' daily routine tasks that consume much time and energy. We believe that the existence of a convenient plagiarism detection instrument can be a noticeable contributing factor for better quality of courses offered at educational institutions. The teachers really need to have more time to be concentrated on developing the course content, creating interesting tasks, or organizing problem-based learning teams, while it is in the interests of our society that the students study harder, honestly, more efficiently and have lower chances to pass the course by copying their peers' work. Some researchers observe that even skillful students often feel demotivated when they see others passing courses using "borrowed" solutions (which happens in large courses, where the teachers have no time for appropriate checks of the whole corpus of student submissions) [8], so an automated plagiarism detection system would have a positive effect on all parties involved in course activities.

III. Plagiarism Prevention and Detection Techniques

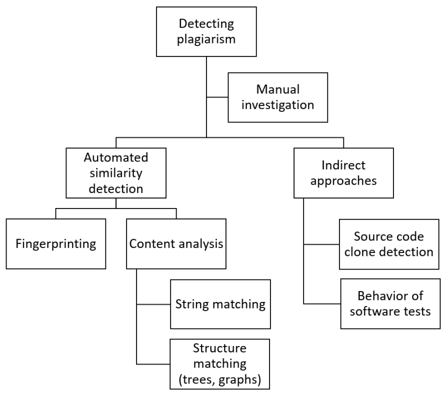
Detecting plagiarism
Manual investigation
Automated similarity detection
Indirect approaches
Fingerprinting
Content analysis
Source code clone detection
String matching
Behavior of software tests
Structure matching (trees, graphs)

Figure 1. Taxonomy of plagiarism detection methods.

It is often noted in the literature that plagiarism prevention should be the main goal of a teacher, while detection is seen as a "last resort" measure. Various methods of plagiarism prevention were suggested. Generally, they are aimed to make plagiarism technically hard, socially unacceptable, and legally dangerous [3]. The teachers are advised to design personalized assignments and conduct onsite problem solving sessions, when possible. School policymakers are expected to devise "honor codes" and similar mechanisms to make the students aware of high importance of "fair play" principles at a given institution. Furthermore, violators of the code are expected to face severe disciplinary punishment.

Nevertheless, plagiarism detection is still seen as a valuable activity, since the very knowledge of plagiarism checking may deter cheating. As noted above, numerous works are dedicated to the technical side of the problem. Even systems belonging to a relatively narrow group of "hermetic detection systems for source code" [9] can be categorized into several classes according to their basic approach to detection [3] (see Figure 1).

Over the last decades, a number of software similarity and source code clone detection methods were developed [3], [10], [11], [12], [13]. However, the evaluation of their relative detection performance, applicability to particular programming languages, and robustness to refactoring techniques is still a subject of research works [5].

In particular, the chosen method(s) should be able to identify matching fragments of the source code files, which is not always possible for techniques based on normalization by decompilation.

IV. Introducing the Project: Sources and Project Goals

The principal challenge of the project is to create a system that would combine high plagiarism detection efficiency with the simplicity of use and appeal to a broad teacher audience by providing capabilities consistent with pedagogical goals and teaching practice.

At least, the following goals should be achieved in a system, applicable in a programming course context:

- "Hermetic" plagiarism detection in a collection of student-submitted assignments. By "hermetic" we mean that borrowings are expected to be found in the same collection rather than in an online source [9].
- Exclusions of certain fragments marked as "templates" by the teacher. Teachers often provide code templates that are expected to be integrated into a student's solution, and these templates should not be considered unauthorized borrowings.
- Simplified detection and reporting procedure "historical" submissions. When students copy from their peers, it is desirable to identify specific pairs of similar documents and deal with their authors on individual basis. However, if a student copied an assignment submitted in the past, it might be enough to provide a simplified report merely proving the fact of cheating.
- Rich reporting and visualization capabilities, which would enable the teachers to find clusters of matching documents efficiently and perform manual analysis of identified similarities.
- Support for a variety of programming languages, and an option to "tokenize" code (see, for example, [14]), which helps to identify plagiarism even if the source code is refactored (lines rearranged, variables renamed, etc.).
- An efficient approximate matching algorithm, able to find partial matches located in non-contiguous areas of source documents.
- Integration with online course management systems (such as Moodle) and/or online code testing tools (such as Aizu online judge [15]).

Some of these capabilities can be are found in earlier systems [16]. For example, WCopyfind [17] for NLP plagiarism detection provided features for HTNL reporting and worked with user-defined document collections. Sherlock [18] supported templates, text tokenization, as well as certain visualization instruments.

V. Conclusion

Since, this paper describes an ongoing project, many issues still need more study and efforts. We need conduct interviews with other programming class teachers to identify all actual classroom practices including the use of online submission systems, current plagiarism prevention and detection measurements, and grading policy. Careful analysis of these practices will help in revealing the practical use cases for plagiarism detection to be used as a basis for our system. In large, achieving the project goals may significantly affect further development of innovative programming teaching practices (such as those described in [19]) within the scope of improving computer education.

Acknowledgement

The work is supported by the University of Aizu Research Funding.

References

[1] T. Kakkonen and M. Mozgovoy, "Students cyber-plagiarism," in Encyclopedia of Cyber Behavior. IGI Global, 2012, pp. 1168–1177.

[2] R. A. Posner et al., The little book of plagiarism. Pantheon, 2007.

[3] M. Mozgovoy, Enhancing computer-aided plagiarism detection. University Of Joensuu Joensuu, 2007.

[4] 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.

[5] C. Ragkhitwetsagul, J. Krinke, and D. Clark, "A comparison of code similarity analysers," Empirical Software Engineering, vol. 23, no. 4, 2018, pp. 2464–2519.

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

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

[8] D. Chuda, P. Navrat, B. Kovacova, and P. Humay, "The issue of (software) plagiarism: A student view," IEEE Transactions on Education, vol. 55, no. 1, 2011, pp. 22–28.

[9] T. Kakkonen and M. Mozgovoy, "Hermetic and web plagiarism detection systems for student essaysan evaluation of the state-of-the-art," Journal of Educational Computing Research, vol. 42, no. 2, 2010, pp. 135–159.

[10] M. Novak, "Review of source-code plagiarism detection in academia," in 2016 39th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO). IEEE, 2016, pp. 796–801.

[11] D. Kılınç, F. Bozyiğit, A. Kut, and M. Kaya, "Overview of source code plagiarism in programming courses," International Journal of Soft Computing and Engineering (IJSCE), vol. 5, no. 2, 2015, pp. 79–85.

[12] M. Akhin and V. Itsykson, "Clone detection: Why, what and how?" in 2010 6th Central and Eastern European Software Engineering Conference (CEE-SECR). IEEE, 2010, pp. 36–42.

[13] A. Sheneamer and J. Kalita, "A survey of software clone detection techniques," International Journal of Computer Applications, vol. 137, no. 10, 2016, pp. 1–21.

[14] M. Ďuračík, E. Kršák, and P. Hrkút, "Current trends in source code analysis, plagiarism detection and issues of analysis big datasets," Procedia engineering, vol. 192, 2017, pp. 136–141.

[15] C. M. Intisar and Y. Watanobe, "Classification of online judge programmers based on rule extraction from self organizing feature map," in 2018 9th International Conference on Awareness Science and Technology (iCAST). IEEE, 2018, pp. 313–318.

[16] M. Mozgovoy, T. Kakkonen, and G. Cosma, "Automatic student plagiarism detection: future perspectives," Journal of Educational Computing Research, vol. 43, no. 4, 2010, pp. 511–531.

[17] L. Bloomfield, "Software to detect plagiarism: Wcopyfind (version 2.6)," 2009.

[18] M. Joy and M. Luck, "Plagiarism in programming assignments," IEEE Transactions on education, vol. 42, no. 2, 1999, pp. 129–133.

[19] E. Pyshkin, "Liberal arts in a digitally transformed world: Revisiting a case of software development education," in Proceedings of the 13th Central & Eastern European Software Engineering Conference in Russia, ser. CEE-SECR '17. New York, NY, USA: ACM, 2017, pp. 23:1–23:7.