Detection of Programming Plagiarism in Computing Education: A Systematic Mapping Study

Kaio Pablo Gomes[1], Simone Nasser Matos[2]

[1]Department of Informatics – Federal University of Technology – Paraná (UTFPR)
Ponta Grossa - PR, Brazil.

[2]Department of Informatics – Federal University of Technology – Paraná (UTFPR)
Ponta Grossa - PR, Brazil.

kgomes@alunos.utfpr.edu.br, snasser@utfpr.edu.br

Abstract. *The programming plagiarism is increasingly a problem in computing education, and the proposed solutions for this growing concern rely on automatic detectors. The usage of the automatic tools for this purpose can provide benefits in education for professors and instructors of programming assignments, besides, to avoid the lack of essential skills from the students since they compromise their programming logic by plagiarizing. This paper performs a systematic mapping study aligned with a snowballing technique to analyzes the existing solutions for this domain. As contributions, tendencies, as well as information analysis, are provided to guide new proposals of solutions.*

1. Introduction

Plagiarism in computing education, specifically in programming, is a problem that compromises critical skills for the development of programming logic by the students [Gomes and Matos 2019]. This misconduct occurs when a person copies either an entire program or a piece of source code made by other people [Andrianov et al. 2020]. As a solution for this academic dishonest, the usage of automatic detectors can reduce plagiarism in student's assignments. Besides, the detection tools can reduce time consumption from professors' grading and increase precision in the plagiarism analysis task [Gomes and Matos 2019].

Specific approaches were created to deal with programming plagiarism by elaborating solutions such as JPLAG, MOSS, Sherlock, SIM, YAP3, and Plaggie. For example, some of the main techniques for this purpose are tokens, hashes, common substrings, and signatures (fingerprints) [Allyson et al. 2018]. As shown in [Xu et al. 2020], these solutions for plagiarism detection still are limited by the demand for practical application. New opportunities and challenges generate the need for different approaches in order to accomplish other requirements. For example, "the capability to detect partial plagiarism," "the resiliency to advanced code obfuscation," "the interpretability of detection results," and "the scalability to process large-scale software."

Through a systematic mapping study (SMS) [Varela et al. 2017], the tendencies and insights about the existing models of plagiarism detection in programming can be analyzed. In order to extend the coverage for proposed detectors, a review technique such as snowballing [Wohlin 2014] assists in the search for complementary references. In this paper, an SMS with the snowballing technique is performed to evaluate solutions that deal with plagiarism in programming, which can be used for computing education.

DOI: 10.5753/cbie.sbie.2020.1633

2. Methodology

The SMS is a specific study that analysis the findings in the literature for a subject of interest [Varela et al. 2017]. The SMS model adopted in this paper is based on the study shown in [Rattan et al. 2013] with modifications related to choosing each analyzed research. A flowchart illustrated in Figure 1 shows the functioning of the performed study, which can be abstracted into eight stages.

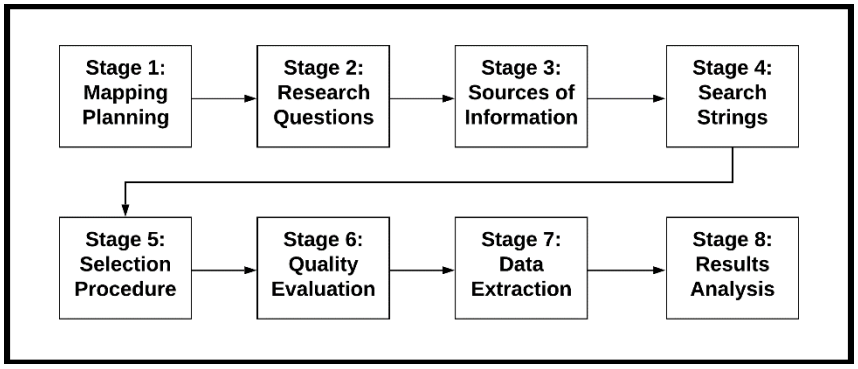

Figure 1. Stages adopted in the systematic mapping study

The first stage establishes SMS conduction guidelines, for example, the coverage period that was defined to contemplate from 2013 to 2018, and the subject to be investigated. From the chosen topic, stage 2 determines the study motivations for creating research questions. The main goals of this SMS were analysis the automatic detectors of plagiarism in programming by identifying their approaches, evaluating processes, supporting languages.

The following three research questions were created: 1) What are the approaches used to identify plagiarism in source code? 2) What plagiarism detection approaches have been evaluated through tests, and how many tests have been conducted? 3) What programming languages are supported in plagiarism detection approaches?

Step 3 considers which sources of information are used for searches. The IEEE Xplore and ACM DL were chosen since they are two of the most relevant in computer science topics, as shown in [Buchinger et al. 2014]. The search strings used in these sources of information represent stage 4, which was elaborated according to Table 1.

From the search results, a selection procedure in stage 5 defines the inclusion and exclusion criteria for choosing each research. The inclusion factors are the type of paper that only were considered those from conferences and journals, and the period of publication as defined in stage 1. The primary factor of exclusion was the presence of duplicated papers in addition to factors based on titles and abstracts out of scope. Altogether, at the end of the selection procedure, approximately 65.97% of the included studies were eliminated by the exclusion criteria.

Table 1. Search strings used in each source of information

Search String	Source of Information
1. ((Source*) OR (Software*) OR (Pro-gram*)) AND (Plagiari*)	IEEE Xplore e ACM DL

The quality evaluation (stage 6) seeks to rank the found studies as a form of identifying the most relevant works in the SMS. The adopted criterion was ordered by the

number of citations, as shown in [Pagani et al. 2015]. After performing the classification, it was identified that 18.18% of the studies have not yet obtained citations. The average citation rate that each work has in this rank is five citations. However, a percentage of 76.32% of citations are concentrated among the top ten.

The data extraction in stage 7 identifies the following information for each research: title, authors, year of publication, type of paper, evaluation process, supported language, and approach. It started at the exclusion phase of the selection process in this paper. The last stage analysis every research in order to answer the three question elaborated in stage 2. The obtained results are discussed in the next chapter.

3. Results

After passing from the first to the seventh stage, the SMS found 33 studies for results analysis in the last stage to answer the three research questions [Chan et al. 2013, Choi et al. 2013, Kim et al. 2013, Tian et al. 2013, Zhang and Liu 2013, Ajmal et al. 2014, Baby et al. 2014, Kikuchi et al. 2014, Lazar and Banias 2014, Liu et al. 2014, Pohuba et al. 2014, Zhang et al. 2014, Acompora and Cosma 2015, Dutta 2015, Jhi et al. 2015, Oprişa and Ignat 2015, Sharma et al. 2015, Soh et al. 2015, Tian et al. 2015, Wang et al. 2015, Ming et al. 2016, Strileţchi et al. 2016, Agrawal and Sharma 2017, Jain et al. 2017, Kargén and Shahmehri 2017, Karnalim 2017, Luo et al. 2017, Mirza et al. 2017, Mišić et al. 2017, Schneider et al. 2017, Sudhamani and Rangarajan 2017, Karnalim 2018, Roopam and Singh 2018].

As a result of the first research question, 26 different approaches were identified. Figure 2 shows the usage frequency of approaches. Note that different works use the same techniques, with the token the most used among all identifications. However, approximately 46.15% of the approaches had the lowest frequency, indicating a lack of further validations. From the functioning of each solution, it was possible to perceive a trend associated with the usage of approaches with more than one technique. Altogether there were 31 studies, which represent 93.94% of the solutions. There was one technique only in two studies representing 6.06% out of all.

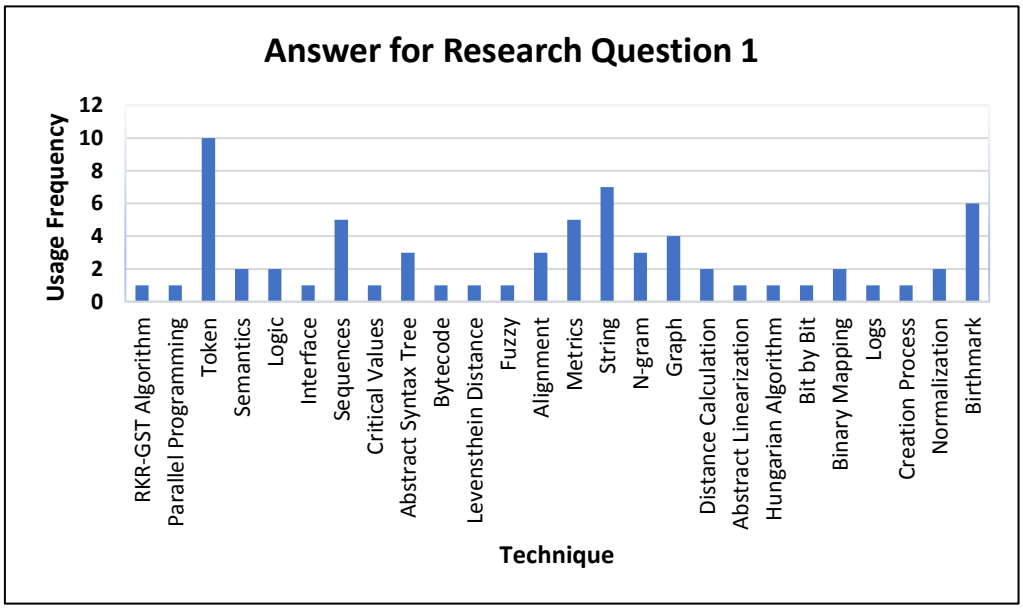

Figure 2. Frequency of the identified approaches

Regarding the research question 2, Figure 3 shows the relation of frequency and number of source codes used in tests. It was found that 96.97% of the works used tests to validate the obtained results, while only one (3.03%) of them [Pohuda et al. 2014] proposed a solution without specifying the evaluation process. Points out that in [Roopam and Singh 2018] did not specify the number of source codes and was considered at least a test for each different repository.

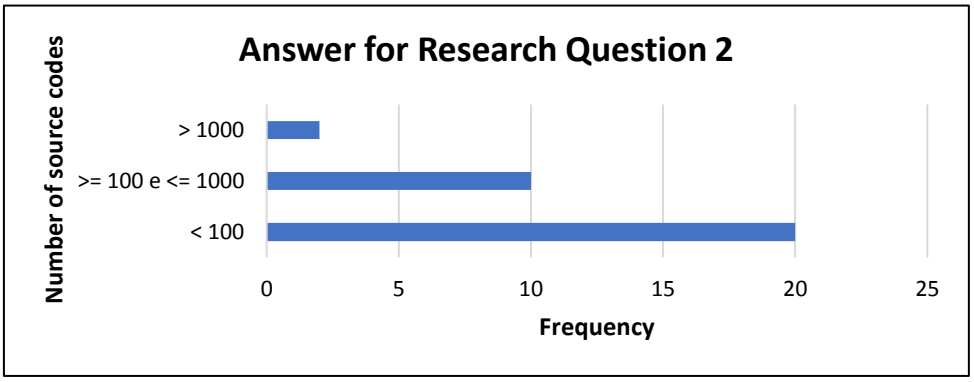

Figure 3. The number of source codes used in tests and its frequency

For the evaluation of the approaches, less than 100 source codes for testing were presented by 60.60% of the studies. The second highest frequency was the usage of between 100 and 1000 source codes, which was adopted by 30.30% out of all. The least frequent process used more than 1000 source codes, and it was chosen by 9.10% of the works.

It not common to share a source code dataset used for testing, only in the research made by [Mirza et al. 2017] adopted this type of resource for the evaluation process. The other works created their tests by developing plagiarism models implemented with the collaboration of students from computing courses. Another identified way of creating tests was generating either automatically or manually samples.

As an answer to research question 3, eight different programming languages were chosen to be supported for the solutions, as shown in Figure 4. However, 39.39% of the studies created independent approaches that do not limit the support feature. These studies show the importance of dealing with different languages, and it has surged a new tendency of using flexible solutions for supporting any source code.

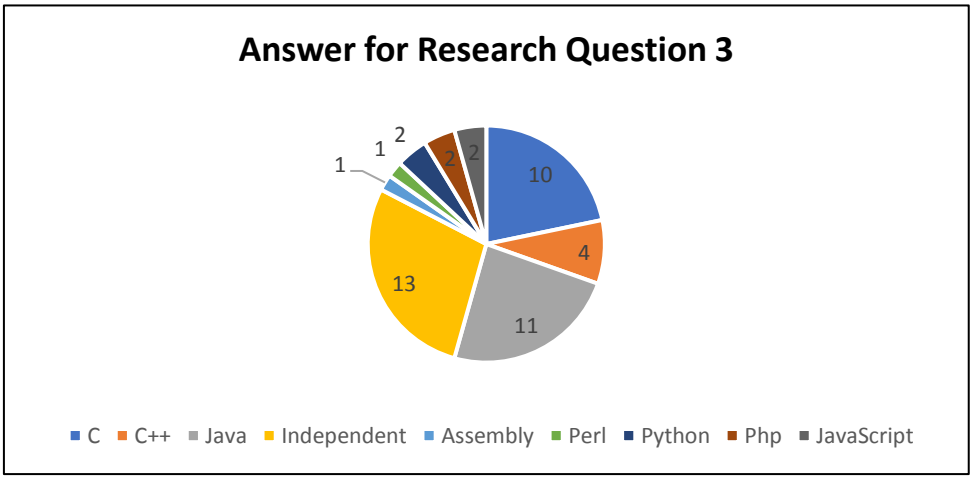

Figure 4. Frequency of supported programming languages

Among the solutions with a limited number of supported programming languages, the usage of Java stands out by representing 11 of 33 studies followed by the C language that is found in 10 of 33. It is identified that these types of dependent approaches tend to support more than one language.

4. Snowballing

The SMS accomplished in this paper was passed through an evaluation to certify whether there were changes in its obtained results considering a most recent coverage period. The snowballing applied to this paper was responsible for contemplating other studies between 2019 and 2020 in order to aggregate content with analysis and updates.

The snowballing technique strictly followed every stage in the SMS again except the two sources of information. Only the most relevant source in the computing between both was considered, which was the IEEE Xplore, as pointed by [Buchinger et al. 2014]. In all, 11 new studies presented in Table 2 were identified as candidates for the SMS.

Table 2. Data from identified studies between 2019 and 2020 with snowballing

Authors	Approach	Number of source codes	Language
[Pham and Nguyen 2019]	K-Grams, RKR algorithm, and Suffix Array	295	C++
[Mahbub et al. 2019]	Support Vector Machine Classifiers, Deep Neural Networks and Random Forests	6063	Language-Independent
[Kim et al. 2019]	Ordered Labeled Tree and LCS algorithm	768	Language-Independent for Android Apps
[Kurtukova et al. 2019]	Machine Learning and Deep neural network	227756	Language-Independent
[Sun et al. 2019]	Bidirectional Static Slicing and Similarity Measures	619	Java
[Herrera et al. 2019]	Metrics	12426	Language-Independent
[Cheers et al. 2019]	Logic	5	Java
[Ljubovic and Pajic 2020]	Machine Learning and Metrics	11388	Language-Independent
[Andrianov et al. 2020]	Sparse Suffix Trees and binary mapping	Not mentioned	Language-Independent
[França et al. 2019]	Normalization and Sherlock N-overlap algorithm	2226	C and Java
[Pajić and Ljubović 2019]	Genetic Algorithm, metrics, and Similarity Measures	Not mentioned	C and C++

By analyzing the results from the applied snowballing, it was noted that most trends identified in the SMS were confirmed in these new studies. Regarding the research question 1, approximately 81.82% of the solutions were based on combined techniques, and only 18.18% used a single type. There were ten different techniques found in the studies in relation to the first SMS, three of which introduced the artificial intelligence

field as a resource for three approaches. Nearly 42.30% of the techniques used in the SMS were reused in nine of the 11 new studies, and the most used approach was based on metrics rather than tokens.

For the new findings from research question 2, it was found that 100% of the works had an evaluation process. However, two of them did not mention the number of source codes used as tests. Among those who mentioned, approximately 88.88% exceeded the most common range of source code quantity in the SMS.

Regarding the supported programming languages, approximately 54.55% of the studies are developing a language-independent approach reiterating to be a desired feature. One of these solutions is aimed only at Android Apps; therefore, it covers any implementation language for this type of application. Still reiterating analyzes from the SMS, for those language-dependent solutions, Java was the most used followed by C and C++.

5. Concluding Remarks

This paper carried out an SMS to analyze the proposed solutions for the programming plagiarism problem in computing education. The applied methodology aligned with a snowballing technique aimed to identify three research questions from the most relevant studies from 2013 to 2020. The questions looked for retrieve information from the approaches, tests, and supported languages of the solutions.

Regarding the identified approaches, it was found the occurring of varied solutions represented by the usage of 36 different techniques in 44 studies. Also, this analysis indicated the presence of combined techniques in approximately 90.90% of the proposals. The evaluation process of these solutions did not occur only in one case, and three studies did not mention the number of source codes for tests. About the highest frequency of source codes used in experiments, it was less than 100 in the first SMS, and more than 100 for studies from 2019 and 2020.

A trend feature was observed by analyzing the research question 3. The feature is the creation of language-independent approaches found in 39.39% of the first SMS and 54.55% on the studies through snowballing. Among those language-dependent solutions, it was identified eight different supported programming languages: C, C++, Java, Assembly, Perl, Python, Php, and JavaScript. The most used language for supporting was Java, followed by C.

As future work, additional research questions can be added to the applied SMS as well as changes in its stages, such as covering a more extended period. For example, it is suggested to investigate whether there is a standard dataset for the evaluation process of the studies to provide conditions for comparing to each other. Besides, the identification of the performance metrics obtained by the solutions such as recall, precision, and f-measure can complement the evaluation process analysis.

Acknowledgment

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – Brasil (CAPES) – Finance Code 001.

References

Acampora, G. and Cosma, G. (2015). A fuzzy-based approach to programming language independent source-code plagiarism detection. In *2015 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*, pages 1–8. IEEE.

Agrawal, M. and Sharma, D. K. (2017). A novel method to find out the similarity between source codes. In *2016 IEEE Uttar Pradesh Section International Conference on Electrical, Computer and Electronics Engineering (UPCON)*, pages 339–343. IEEE.

Ajmal, O., Missen, M. S., Hashmat, T., Moosa, M., and Ali, T. (2014). Eplag: A two layer source code plagiarism detection system. In *Eighth International Conference on Digital Information Management (ICDIM 2013)*, pages 256–261. IEEE.

Andrianov, I., Rzheutskaya, S., Sukonschikov, A., Kochkin, D., Shvetsov, A., and Sorokin, A. (2020). Duplicate and plagiarism search in program code using suffix trees over compiled code. In *2020 26th Conference of Open Innovations Association (FRUCT)*, pages 1–7.

Baby, J., Kannan, T., Vinod, P., and Gopal, V. (2014). Distance indices for the detection of similarity in c programs. In *2014 International Conference on Computation of Power, Energy, Information and Communication (ICCPEIC)*, pages 462–467. IEEE.

Buchinger, D., de Siqueira Cavalcanti, G. A., and da Silva Hounsell, M. (2014). Mecanismos de busca acadêmica: uma análise quantitativa. Revista Brasileira de Computação Aplicada, 6(1):108–120.

Chan, P. P., Hui, L. C., and Yiu, S.-M. (2013). Heap graph based software theft detection. IEEE Transactions on Information Forensics and Security, 8(1):101–110.

Cheers, H., Lin, Y., and Smith, S. P. (2019). A novel approach for detecting logic similarity in plagiarised source code. In *2019 IEEE 10th International Conference on Software Engineering and Service Science (ICSESS)*, pages 1–6.

Choi, J., Han, Y., Cho, S.-j., Yoo, H., Park, M., Han, S., You, I., and Song, I. (2013). A survey of feature extraction techniques to detect the theft of windows applications. In *2013 Seventh International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing*, pages 723–728. IEEE.

Dutta, R. (2015). Efficient approach to detect logical equivalence in the paradigm of software plagiarism. In *Proceedings of the 2015 Third International Conference on Computer, Communication, Control and Information Technology (C3IT)*, pages 1–5. IEEE.

França, A. B., Maciel, D. L., Soares, J. M., and Barroso, G. C. (2019). Sherlock n-overlap: Invasive normalization and overlap coefficient for the similarity analysis between source code. *IEEE Transactions on Computers*, 68(5):740–751.

Gomes, K. P. and Matos, S. (2019). Contributions of bioinformatics for computing education in the detection of programming assignment plagiarism. In *Brazilian Symposium on Computers in Education (Simpósio Brasileiro de Informática na Educação-SBIE)*, volume 30, page 1351.

Herrera, G., Nuñez-del-Prado, M., Lazo, J. G. L., and Alatrista, H. (2019). Through an agnostic programming languages methodology for plagiarism detection in engineering

coding courses. In *2019 IEEE World Conference on Engineering Education (EDU-NINE)*, pages 1–6.

Jain, S., Kaur, P., Goyal, M., and Dhanalekshmi, G. (2017). Cplag: Efficient plagiarism detection using bitwise operations. In *2017 Tenth International Conference on Contemporary Computing (IC3)*, pages 1–5. IEEE.

Jhi, Y. C., Jia, X., Wang, X., Zhu, S., Liu, P., and Wu, D. (2015). Program characterization using runtime values and its application to software plagiarism detection. *IEEE Transactions on Software Engineering*, 41(9):925–943.

Kargén, U. and Shahmehri, N. (2017). Towards robust instruction-level trace alignment of binary code. In *2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 342–352. IEEE.

Karnalim, O. (2017). Detecting source code plagiarism on introductory programming course assignments using a bytecode approach. In *2016 International Conference on Information & Communication Technology and Systems (ICTS)*, pages 63–68. IEEE.

Karnalim, O. (2018). An abstract method linearization for detecting source code plagiarism in object-oriented environment. In *2017 8th IEEE International Conference on Software Engineering and Service Science (ICSESS)*, pages 58–61. IEEE.

Kikuchi, H., Goto, T., Wakatsuki, M., and Nishino, T. (2014). A source code plagiarism detecting method using alignment with abstract syntax tree elements. In *15th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD)*, pages 1–6. IEEE.

Kim, B., Lim, K., Cho, S. J., and Park, M. (2019). Romadroid: A robust and efficient technique for detecting android app clones using a tree structure and components of each app's manifest file. *IEEE Access*, 7:72182–72196.

Kim, Y., Moon, J., Kim, D., Jeong, Y., Cho, S. J., Park, M., and Han, S. (2013). A static birthmark of windows binary executables based on strings. In *2013 Seventh International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing*, pages 734–738. IEEE.

Kurtukova, A., Romanov, A., and Fedotova, A. (2019). De-anonymization of the author of the source code using machine learning algorithms. In *2019 International Multi-Conference on Engineering, Computer and Information Sciences (SIBIRCON)*, pages 0612–0617. IEEE.

Lazar, F. M. and Banias, O. (2014). Clone detection algorithm based on the abstract syntax tree approach. In *2014 IEEE 9th IEEE International Symposium on Applied Computational Intelligence and Informatics (SACI)*, pages 73–78. IEEE.

Liu, K., Zheng, T., and Wei, L. (2014). A software birthmark based on system call and program data dependence. In *2014 11th Web Information System and Application Conference*, pages 105–110. IEEE.

Ljubovic, V. and Pajic, E. (2020). Plagiarism detection in computer programming using feature extraction from ultra-fine-grained repositories. *IEEE Access*, 8:96505–96514.

Luo, L., Ming, J., Wu, D., Liu, P., and Zhu, S. (2017). Semantics-based obfuscation-resilient binary code similarity comparison with applications to software and algorithm plagiarism detection. *IEEE Transactions on Software Engineering*, 43(12):1157–1177.

Mahbub, P., Oishie, N. Z., and Haque, S. R. (2019). Authorship identification of source code segments written by multiple authors using stacking ensemble method. In *2019 22nd International Conference on Computer and Information Technology (ICCIT)*, pages 1–6. IEEE.

Ming, J., Zhang, F., Wu, D., Liu, P., and Zhu, S. (2016). Deviation-based obfuscation-resilient program equivalence checking with application to software plagiarism detection. *IEEE Transactions on Reliability*,65(4):1647–1664.

Mirza, O. M., Joy, M., and Cosma, G. (2017). Style analysis for source code plagiarism detection—an analysis of a dataset of student coursework. In *2017 IEEE 17th international conference on advanced learning technologies (ICALT)*, pages 296–297. IEEE.

Mišić, M. J., Protić, J. Ž., and Tomašević, M. V. (2017). Improving source code plagiarism detection: lessons learned. In *2017 25th Telecommunication Forum (TELFOR)*, pages 1–8. IEEE.

Opris¸a, C. and Ignat, N. (2015). A measure of similarity for binary programs with a hierarchical structure. In *2015 IEEE International Conference on Intelligent Computer Communication and Processing (ICCP)*, pages 117–123. IEEE.

Pagani, R. N., Kovaleski, J. L., and Resende, L. M. (2015). Methodi ordinatio: a proposed methodology to select and rank relevant scientific papers encompassing the impact factor, number of citation, and year of publication. *Scientometrics*, 105(3):2109–2135.

Pajić, E. and Ljubović, V. (2019). Improving plagiarism detection using genetic algorithm. In *2019 42nd International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, pages 571–576.

Pham, M. T. and Nguyen, T. B. (2019). The domjudge based online judge system with plagiarism detection. In *2019 IEEE-RIVF International Conference on Computing and Communication Technologies (RIVF)*, pages 1–6. IEEE.

Pohuba, D., Dulík, T., and Jank, P. (2014). Automatic evaluation of correctness and originality of source codes. In *10th European Workshop on Microelectronics Education (EWME)*, pages 49–52. IEEE.

Rattan, D., Bhatia, R., and Singh, M. (2013). Software clone detection: A systematic review. *Information and Software Technology*, 55(7):1165–1199.

Roopam and Singh, G. (2018). To enhance the code clone detection algorithm by using hybrid approach for detection of code clones. In *2017 International Conference on Intelligent Computing and Control Systems (ICICCS)*, pages 192–198.

Schneider, J., Bernstein, A., Vom Brocke, J., Damevski, K., and Shepherd, D. C. (2017). Detecting plagiarism based on the creation process. *IEEE Transactions on Learning Technologies*, 11(3):348–361.

Sharma, S., Sharma, C. S., and Tyagi, V. (2015). Plagiarism detection tool "parikshak". In *2015 International Conference on Communication, Information & Computing Technology (ICCICT)*, pages 1–7. IEEE.

Soh, C., Tan, H. B. K., Arnatovich, Y. L., and Wang, L. (2015). Detecting clones in android applications through analyzing user interfaces. In *2015 IEEE 23rd international conference on program comprehension*, pages 163–173. IEEE.

Strileţchi, C., Vaida, M., Chiorean, L., and Popa, S. (2016). A cross-platform solution for software plagiarism detection. In *2016 12th IEEE International Symposium on Electronics and Telecommunications (ISETC)*, pages 141–144. IEEE.

Sudhamani, M. and Rangarajan, L. (2017). Code clone detection based on order and content of control statements. In *2016 2nd International Conference on Contemporary Computing and Informatics (IC3I)*, pages 59–64. IEEE.

Sun, W., Wang, X., Wu, H., Duan, D., Sun, Z., and Chen, Z. (2019). Maf: method-anchored test fragmentation for test code plagiarism detection. In *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering Education and Training (ICSE-SEET)*, pages 110–120. IEEE.

Tian, Z., Zheng, Q., Liu, T., and Fan, M. (2013). Dkisb: Dynamic key instruction sequence birthmark for software plagiarism detection. In *2013 IEEE 10th International Conference on High Performance Computing and Communications & 2013 IEEE International Conference on Embedded and Ubiquitous Computing*, pages 619–627. IEEE.

Tian, Z., Zheng, Q., Liu, T., Fan, M., Zhuang, E., and Yang, Z. (2015). Software plagiarism detection with birthmarks based on dynamic key instruction sequences. *IEEE Trans. Software Eng.*, 41(12):1217–1235.

Varela, A. S. N., Pérez-Gonzalez, H. G., Martínez-Perez, F. E., and Soubervielle-Montalvo, C. (2017). Source code metrics: A systematic mapping study. *Journal of Systems and Software*, 128:164–197.

Wang, H., Zhong, J., and Zhang, D. (2015). A duplicate code checking algorithm for the programming experiment. In *2015 Second International Conference on Mathematics and Computers in Sciences and in Industry (MCSI)*, pages 39–42. IEEE.

Wohlin, C. (2014). Guidelines for snowballing in systematic literature studies and a replication in software engineering. In *Proceedings of the 18th international conference on evaluation and assessment in software engineering*, pages 1–10.

Xu, X., Fan, M., Jia, A., Wang, Y., Yan, Z., Zheng, Q., and Liu, T. (2020). Revisiting the challenges and opportunities in software plagiarism detection. In *2020 IEEE 27th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 537–541. IEEE.

Zhang, F., Wu, D., Liu, P., and Zhu, S. (2014). Program logic based software plagiarism detection. In *2014 IEEE 25th International Symposium on Software Reliability Engineering*, pages 66–77. IEEE.

Zhang, Li, P. and Liu, Dong, S. (2013). Ast-based multi-language plagiarism detection method. In *2013 IEEE 4th International Conference on Software Engineering and Service Science*, pages 738–742. IEEE.