# Work-in-Progress: Syntactic Code Similarity Detection in Strongly Directed Assessments

*by* Oscar Karnalim, Simon, Mewati Ayub Gisela Kurniawati, Dkk,.

# Work-in-Progress: Syntactic Code Similarity Detection in Strongly Directed Assessments

Oscar Karnalim*†, Simon*, Mewati Ayub†, Gisela Kurniawati†, Rossevine Artha Nathasya†
and Maresha Caroline Wijanto†
*College of Engineering, Science, and Environment, University of Newcastle, Callaghan, Australia
oscar.karnalim@uon.edu.au, simon@newcastle.edu.au
†Faculty of Information Technology, Maranatha Christian University, Bandung, Indonesia
{mewati.ayub, gisela.kurniawati, rossevine.an, maresha.cw}@it.maranatha.edu

***Abstract*—When checking student programs for plagiarism and collusion, many similarity detectors aim to capture semantic similarity. However, they are not particularly effective for strongly directed assessments, in which the student programs are expected to be semantically similar. A detector focusing on syntactic similarity might be useful, and this paper reports its effectiveness on programming assessment tasks collected from algorithms and data structures courses in one academic semester. Our study shows that syntactic similarity detection is more effective than its semantic counterpart in strongly directed assessments, with some irregular similarity patterns being useful for raising suspicion. We also tested whether take-home assessments have higher similarity than in-class assessments, and confirmed that hypothesis. Consistency of the findings will be further validated on other courses with strongly directed assessments, and a syntactic similarity detector specifically tailored for strongly directed assessments will be proposed.**

***Index Terms*—syntactic similarity, programming, assessment, plagiarism, collusion**

## I. Introduction

Some students might be tempted to cheat – an act that has been characterised as a desperate act of help-seeking [1] – if the potential benefits far outweigh the drawbacks and they have an opportunity to do so [2]. Two of the more common forms of cheating in programming are plagiarism and collusion [3]. Both are about illegitimately reusing other people's programs; the difference is in whether the authors of the original work are involved in the copying, as is generally the case with collusion.

Detecting plagiarism and collusion in programming is often aided by a similarity detection tool such as JPlag [4] or MOSS [5]. First the tool reports program pairs that display undue similarity, then the pairs are examined by the lecturer to determine whether they are suspicious [6], [7].

Each time suspicion is raised, it needs to be supported by sufficient evidence, demonstrating that the similarity has not arisen by coincidence [8]. Typically, the unduly similar segments of the suspected programs would represent irregular patterns such as unusual surface structure, program flow, or use of libraries.

At this point it will be helpful to distinguish three levels of similarity that a pair of programs might display: surface, syntactic, and semantic similarity. They are derived from Roy et al.'s classification of code clones [9]. Surface similarity occurs when two code segments are verbatim or they differ only in terms of comments and/or white space (type I code clone). Syntactic similarity occurs when two code segments share the same program statements with variations in identifiers, string literals, data types, comments, and white space (type II code clone). Semantic similarity occurs when two code segments have the same functionality regardless of variation in the program statements and/or syntax (type III and type IV code clone). We do not directly use Roy et al.'s classification since the terms are less self-explanatory for computing educators.

Most code similarity detection tools are designed to capture semantic similarity, assuming that each assessment task can have many possible semantically distinct solutions. Before checking for semantic similarity, the tools will typically nullify non-semantic variation: for example, by generalising identifier names [4], removing string constants [10], replacing program module calls with the code of the module's body [11], or replacing combined variable declarations with multiple individual variable declarations [12]. Some of the tools even conduct the program comparisons on direct semantic representations such as program dependency graphs [13] or executable program binaries [14].

However, not all programming assessments are open to multiple semantically distinct solutions, especially in early programming courses when most aspects of the solutions are semantically similar by default [15]. Many of those are either trivial and/or strongly directed. Trivial assessments are expected to be easy and can be completed in a short time. Strongly directed assessments require the students to satisfy constraints such as following a particular structure, and this limits the variation among solutions.

There are at least three differences between these types of assessment. First, strongly directed assessments are not always trivial, and vice versa. For example, implementing a heap in an array is strongly directed but might be challenging if students are only given a high-level algorithm that regards the heap as a tree. Second, strongly directed assessments often require a higher level of prior programming knowledge than trivial assessments as some of the assessment constraints are quite technical. Third, strongly directed assessments typically give rise to longer solutions than trivial assessments.

Imposing the use of semantic similarity detection tools on those assessments might result in all programs being suspected of plagiarism or collusion as they share the same semantics. The report is unhelpful to the lecturer, who then needs to examine all program pairs, with many of the similarities unlikely to be evidence of misconduct.

In response to this issue, some detection tools focus on syntactic and/or surface similarity. Kustanto and Liem [16] and Sulistiani and Karnalim [17], for example, propose detection tools that consider syntactic variation in measuring program similarity; their approach is simply to tokenise student programs prior to comparison, with no further preprocessing. This approach is partly followed by Inoue and Wada [18], whose approach considers comments, white space, and unusual code patterns. Joy and Luck [8] propose a tool that can compare student programs based on multiple representations, starting from the original textual forms and culminating in token strings in which some of the tokens are generalised based on their corresponding types. Karnalim and Simon [19] and Nichols et al. [20] propose detection tools that capture more syntactic information by comparing student programs based on their syntax trees and parse trees respectively.

This paper compares the effectiveness of a syntactic and a semantic similarity detection tool for detecting plagiarism and collusion in strongly directed assessments, which were collected from two algorithms and data structures courses (basic and advanced) in one academic semester. Existing studies have only conducted such comparisons on trivial assessments. We also report any irregular patterns that are useful for raising suspicion in such assessments in addition to syntax similarity, and this is expected to enrich the perspective of computing lecturers in investigating undue similarity among student programs.

Programming collusion (and probably plagiarism) might arise more frequently in take-home than in lab assessments since take-home assessments are typically less supervised [21], which might expose the students to more temptation to cheat. Since code similarity is often used as the basis for raising suspicion of academic misconduct [6], we hypothesise that designing programming assessments as homework might result in higher levels of similarity, and we test the hypothesis in our data set by comparing each take-home assessment task with a comparable assessment completed in a lab (in a previous offering of the course) in terms of average degree of similarity. To the best of our knowledge, this hypothesis has not been tested in existing studies.

This study thus covers three research questions:

- RQ1 Is syntactic similarity detection more effective than semantic similarity detection in strongly directed assessment tasks?
- RQ2 Which irregular patterns are useful for raising suspicion in strongly directed assessment tasks?
- RQ3 Do take-home assessment tasks result in higher similarity across student submissions than supervised lab assessment tasks?

## II. Method

The three research questions were addressed based on student programs collected at Maranatha Christian University, Indonesia, from two algorithms and data structures courses: basic and advanced, both offered in the first semester of 2020. In total, there are 1034 student programs collected from 55 undergraduate students (age 18 to 20) over 69 programming assessment tasks.

Basic algorithms and data structures (BDS) is a compulsory course for information technology undergraduate students offered in the second semester of their study. The course mainly covers linear data structures and some sorting algorithms. It is available only to students who have passed introductory programming. Two classes of the basic algorithms and data structures course are considered in this study: class A, with 29 students, and class B, with 17 students. While these classes cover the same course material, they are taught independently by different teachers, and so are worth considering as distinct offerings. One to three Python programming assessments were given each week, with a total of 20 assessments in each class over one academic semester. Initially, all assessments were designed to be completed in 100 minutes in a laboratory with direct supervision by the lecturer and the tutors. However, due to an unprecedented global health event, from the sixth week of semester the assessments were issued online as homework and were to be completed within a day.

Advanced algorithms and data structures (ADS) is another compulsory course for information technology undergraduate students, offered in the third semester of their study. As it continues the series of algorithms and data structures courses, covering non-linear data structures, it is available only to students who have taken the BDS course. One ADS class is considered in this study, with only nine students enrolled. One to three programming assessments were given weekly, with a total of 29 assessments over the semester. Most assessments (25) were to be completed with the help of an unpublished code generation tool that facilitates a smooth transition from Python, the first language taught, to Java or C#, programming languages in subsequent courses. The tool introduces two non-Python concepts: variable declaration and non-white-space delimiters (braces and semicolons). In each assessment, the tool would generate Java template code, which students were then to complete with generalised syntax from both Java and C#. The remaining four assessments were to be completed entirely in Java or C# to give the students more experience with their future programming languages.

The first research question (RQ1), whether syntactic similarity detection is more effective than semantic detection in strongly directed assessments, was addressed by comparing the performance of a syntactic similarity detector with that of JPlag [4], a tool that detects semantic similarity via the application of program statement generalisation.

To make the detectors comparable, the syntactic similarity detector is derived from JPlag by excluding program statement generalisation. The detector works in four stages. First, student programs are converted to token strings with ANTLR [22], a process that includes removing comments and white space. Second, tokens representing identifiers, string literals, constants, and numeric data types are generalised to a form that is based on their corresponding type. Third, the token strings are pairwise compared using running Karp-Rabin greedy string tiling [23]. Finally, the program pairs are sorted according to their average similarity [4].

Both JPlag and the syntactic similarity detector are set to have two as the minimum matching length and 75% as the minimum similarity threshold for suspicion, as suggested by Karnalim and Simon [19]. Their effectiveness was measured with top-*k* precision and average degree of similarity of program pairs.

Top-*k* precision, a metric from the field of information retrieval [24], is the proportion of copied program pairs that appear in the top-*k* suspected program pairs. In our context, *k* is the number of program pairs that are actually copied for an assessment task. Precision is often considered in conjunction with recall, the proportion of program pairs that are both copied and suspected to all copied program pairs. However, by this definition, top-*k* recall would be defined by exactly the same formula as top-*k* precision.

Copied program pairs for each assessment were selected by the laboratory lecturer via observation of a pairwise similarity report generated by the syntactic similarity detector. The investigation started from program pairs with the highest similarity, followed by other pairs in descending order. The lecturer deemed a program pair to be suspicious via a complex reasoning process involving task difficulty, expected code variance and length, and whether the code can be readily copied from acceptable sources such as lecture material. We are aware that this selection mechanism is not guaranteed to detect all copied programs. However, it is more time efficient and less labour intensive than entirely manual checking. For each assessment task, the lecturer would need to manually check up to 406 program pairs for BDS class A, up to 136 program pairs for BDS class B, and up to 36 program pairs for ADS. JPlag was not used to select the copied program pairs since our initial investigation shows that the tool reports many program pairs as verbatim copies although they are syntactically different.

Out of 69 assessment tasks, 44 are considered in this study. Twenty-three have no copied program pairs, and so cannot be used to measure top-*k* precision as the formula would have a denominator of zero. Two more required the use of C#, a language that is not covered by our syntactic similarity detection tool.

Syntactic similarity detection is expected to have higher top-*k* precision in strongly directed assessments, but lower average degree of similarity as it does not employ program statement generalisation. We measured the average precision of all assessment tasks within the data, and the average degree of similarity of all program pairs in the data set. The statistical significance of each metric was tested with a two-tailed paired t-test with 95% confidence rate.

The second research question (RQ2), listing useful irregular patterns for raising suspicion in strongly directed assessment tasks, was addressed by asking the laboratory lecturers to record any interesting patterns while selecting the copied program pairs of each assessment task. The patterns were then filtered and summarised by the authors.

The third research question (RQ3), whether take-home assessments result in higher similarity across student submissions than lab-based assessments, was addressed by comparing all take-home assessments from our data set (week 6 onward) to another set of assessments addressing the same tasks but completed in labs, taken from past offerings of the courses. A past offering of BDS had four classes with a total of 59 students. A past offering of ADS had one class with 16 students. We used the syntactic similarity detector to measure the average degree of similarity of the assessment tasks from the past and current offerings. We were not able to measure precision for the past offerings because we have no information about which program pairs were copied.

Statistical significance tests with 95% confidence rate were performed for each comparison at the level of individual assessment tasks. Since the comparison involves two different data sets, the t-test used was selected according to whether the data set variances were equal or unequal.

## III. Results and Discussion

### A. Comparing syntactic to semantic similarity detection

This section compares syntactic to semantic similarity detection in terms of their effectiveness in strongly directed assessments. Two metrics are considered: top-*k* precision and average degree of similarity.

Figure 1 shows that syntactic similarity detection is more effective than semantic similarity detection as performed by JPlag. On most tasks, syntactic detection results in at least the same top-*k* precision, as the delta value (syntactic top-*k* precision – semantic top-*k* precision) is non-negative. On average, the syntactic top-*k* precision (82%) is higher by 18% and the difference is statistically significant ($p<0.001$). This is expected as semantic similarity detection ignores syntactic variation, leading to some difficulties in distinguishing copied pairs as most of the programs are expected to be semantically similar in strongly directed assessments.

Having nullified syntactic variation, semantic similarity detection results in higher degree of similarity (Table I). Although the difference is statistically significant ($p<0.001$), it is not substantial in our data set (less than 1% on average). This is expected since many of the assessment tasks imposed a particular class and method structure, and that resulted in larger proportion of similar code segments. Further, syntactic variation was applicable only to some parts of the solution.

### B. Useful irregular patterns for raising suspicion

This section lists irregular patterns that might be useful for raising suspicion as they are unlikely to be coincidental. The patterns are based on the copied program pairs in our data set, as identified by the laboratory lecturers.

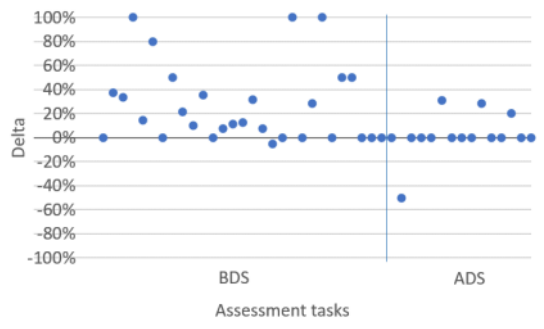

Fig. 1. Difference in top-$k$ precision between syntactic and semantic detection on assessment tasks, calculated by subtracting semantic top-$k$ precision from syntactic top-$k$ precision

TABLE I
DEGREES OF SIMILARITY FOR SYNTACTIC AND SEMANTIC SIMILARITY DETECTION ON ALL PROGRAM PAIRS (9468)

| | Syntactic detection | Semantic detection |
|---|---|---|
| Average | 81.4% | 82.3% |
| Variance | 2.5% | 2.7% |

The most obvious pattern occurs when one program is a verbatim copy of another at surface level, including identifier names, white space, and/or comments. However, this pattern is applicable only when the expected solution is not trivial and most of the program contents are neither automatically generated nor copiable from legitimate sources.

Some shared similarities become irregular and suspicious when they are uncommon and probably out of context at surface level. In an assessment task, for example, a number of students used the unexpected name *bigpower* for an instance of the class *power*.

Irregular patterns can also arise from incompleteness or over-completeness of student programs. If some similar programs do not cover one or more task requirements that are easy to fulfill, they are suspicious. For instance, a few copied programs have the same distinct output which is different from that specified by the task. The suspicion also applies when some similar programs have additional features that are not relevant to the task requirements, such as unnecessary program statements or unused methods.

Although this is rare, it is interesting to see that some perpetrators neglect to remove the original author's identity in the copied program, and this will clearly be evidence for suspicion of collusion. In two assessment tasks, a few perpetrators changed the file names but neglected to change the student IDs within the copied files. In three other assessment tasks, another group of perpetrators failed to rename the project directory, whose name is based on the student ID.

### *C. Comparing take-home to lab assessments*

This section compares average similarity degree of take-home assessments with that of lab assessments. Each considered assessment task is given an ID by concatenating the course ID (BDS or ADS) with a chronological sequence number. For example, the third considered assessment task from the Basic Algorithms and Data Structures is given an ID of BDS03.

On average, BDS take-home assessments have an average degree of similarity that is 4% higher than that of lab assessments. Figure 2 plots the comparison for each assessment task, showing that the increases are all significant; the populations share unequal variances.

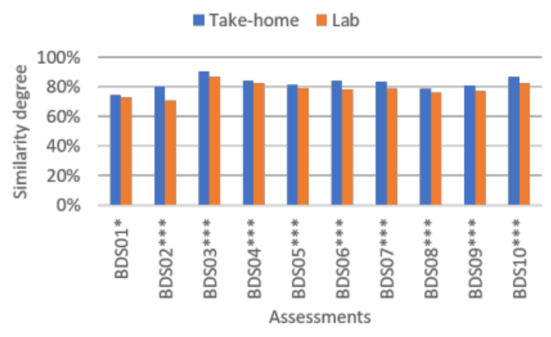

Fig. 2. Average degrees of similarity of take-home and lab assessments in BDS; a single asterisk indicates $p<0.05$ and three asterisks indicate $p<0.001$

As seen in Figure 3, take-home assessments also result in higher degrees of similarity in ADS, where the average increase is two and half times greater (10%) than that in BDS. The larger increase is expected since the assessments in ADS have more copied program pairs than those in BDS while having fewer total programs. The increase is statistically significant on all assessments except ADS02 and ADS13, the two in which the lab assessment results in a higher average degree of similarity.

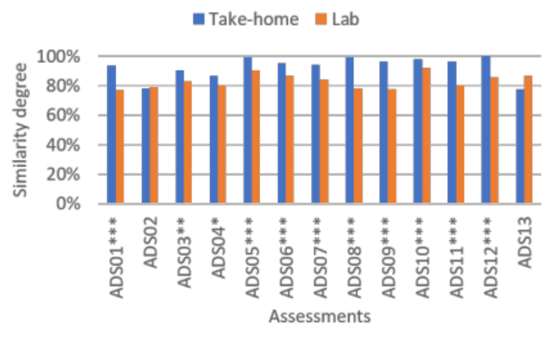

Fig. 3. Average degrees of similarity of take-home and lab assessments in ADS; a single asterisk indicates $p<0.05$; two asterisks, $p<0.01$; and three asterisks, $p<0.001$

To summarise, in our data set, take-home assessments have more code similarity than lab assessments, and thus give rise to more suspicion of plagiarism and/or collusion. This might be taken into consideration when designing subsequent assessments. More comprehensive prevention and detection strategies might be required for take-home assessments, as the code similarities might be the result of academic misconduct.

It is possible that the differences might be simply a result of comparing two different cohorts of students, one of which has more academic misconduct, although the data are drawn from the same courses offered in the same major at the same university. Further investigation is required to test this possibility.

## IV. Conclusion

This paper reports the effectiveness of syntactic similarity detection in strongly directed assessments. According to our study, the use of syntactic code similarity detection is suggested as an alternative to semantic similarity detection since the former outperforms the latter by 18% in terms of precision. Suspicion can be strengthened by observation of some irregular patterns including verbatim copying, irregular surface similarities, incompleteness or over-completeness of the submitted programs, and inadequate de-identification.

Our study also shows that take-home assessments result in higher similarity than lab assessments. Computing lecturers should consider introducing more comprehensive prevention and detection strategies for take-home assessments, as the higher degree of similarity might be the result of breaches of academic integrity.

Possible threats to the validity of our conclusions include the limitation of our study to three classes at a single institution, and the comparison of two different cohorts in those courses when comparing take-home and lab assessment tasks. With regard to the latter point, it is possible that the differences we have found merely indicate that the more recent cohort includes more students who are inclined to copy from one another.

The study is not yet complete since only algorithms and data structures courses are considered. We plan to validate the consistency of our findings on other courses with strongly directed assessments, such as object-oriented programming and application-oriented programming. We also plan to propose a syntactic similarity detector that is specifically tailored for strongly directed assessments. Finally, we intend to further investigate whether take-home assessments result in higher degrees of similarity than lab assessments within the same cohort of students.

## Acknowledgment

To Australia Awards Scholarship for supporting the study of the first author. To Risal from Maranatha Christian University, Indonesia, for partly facilitating the study. To William Chivers from University of Newcastle, Australia, for his contribution to the overall project.

## References

[1] A. Hellas, J. Leinonen, and P. Ihantola, "Plagiarism in take-home exams: help-seeking, collaboration, and systematic cheating," in *22nd Conference on Innovation and Technology in Computer Science Education*, 2017, pp. 238–243.

[2] Simon, "Assessment in online courses: some questions and a novel technique," in *Higher Education in a Changing World: Research and Development in Higher Education*, 2005, pp. 500–506.

[3] R. Fraser, "Collaboration, collusion and plagiarism in computer science coursework," *Informatics in Education*, vol. 13, no. 2, pp. 179–195, 2014.

[4] L. Prechelt, G. Malpohl, and M. Philippsen, "Finding plagiarisms among a set of programs with JPlag," *Journal of Universal Computer Science*, vol. 8, no. 11, pp. 1016–1038, 2002.

[5] S. Schleimer, D. S. Wilkerson, and A. Aiken, "Winnowing: local algorithms for document fingerprinting," in *International Conference on Management of Data*, 2003, pp. 76–85.

[6] O. Karnalim, Simon, and W. Chivers, "Similarity detection techniques for academic source code plagiarism and collusion: a review," in *IEEE International Conference on Engineering, Technology and Education*. IEEE, 2019.

[7] 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*, vol. 19, no. 3, pp. 27:1–27:37, 2019.

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

[9] C. K. Roy, J. R. Cordy, and R. Koschke, "Comparison and evaluation of code clone detection techniques and tools: a qualitative approach," *Science of Computer Programming*, vol. 74, no. 7, pp. 470–495, 2009.

[10] J. Y. Poon, K. Sugiyama, Y. F. Tan, and M.-Y. Kan, "Instructor-centric source code plagiarism detection and plagiarism corpus," in *17th Conference on Innovation and Technology in Computer Science Education*, 2012, pp. 122–127.

[11] J.-H. Ji, G. Woo, and H.-G. Cho, "A source code linearization technique for detecting plagiarized programs," in *12th Conference on Innovation and Technology in Computer Science Education*, 2007, pp. 73–77.

[12] Z. Đurić and D. Gašević, "A source code similarity system for plagiarism detection," *Computer Journal*, vol. 56, no. 1, pp. 70–86, 2013.

[13] H. Cheers, Y. Lin, and S. P. Smith, "Detecting pervasive source code plagiarism through dynamic program behaviours," in *22nd Australasian Computing Education Conference*, 2020, pp. 21–30.

[14] O. Karnalim and S. Budi, "The effectiveness of low-level structure-based approach toward source code plagiarism level taxonomy," in *Sixth International Conference on Information and Communication Technology*. IEEE, 2018, pp. 130–134.

[15] S. Mann and Z. Frew, "Similarity and originality in code: plagiarism and normal variation in student assignments," in *Eighth Australasian Computing Education Conference*, 2006, pp. 143–150.

[16] C. Kustanto and I. Liem, "Automatic source code plagiarism detection," in *10th ACIS International Conference on Software Engineering, Artificial Intelligences, Networking and Parallel/Distributed Computing*, 2009, pp. 481–486.

[17] L. Sulistiani and O. Karnalim, "ES-Plag: efficient and sensitive source code plagiarism detection tool for academic environment," *Computer Applications in Engineering Education*, vol. 27, no. 1, pp. 166–182, 2019.

[18] U. Inoue and S. Wada, "Detecting plagiarisms in elementary programming courses," in *Ninth International Conference on Fuzzy Systems and Knowledge Discovery*, 2012, pp. 2308–2312.

[19] O. Karnalim and Simon, "Syntax trees and information retrieval to improve code similarity detection," in *22nd Australasian Computing Education Conference*, 2020, p. 48–55.

[20] L. Nichols, K. Dewey, M. Emre, S. Chen, and B. Hardekopf, "Syntax-based improvements to plagiarism detectors and their evaluations," in *24th Conference on Innovation and Technology in Computer Science Education*, 2019, pp. 555–561.

[21] O. Karnalim, G. Kurniawati, S. F. Sujadi, and R. A. Nathasya, "Comparing the impact of programming assessment type: in-class vs take-home," *International Journal of Engineering Pedagogy*, vol. 10, no. 4, pp. 125–132, 2020.

[22] T. Parr, *The Definitive ANTLR 4 Reference*. Pragmatic Bookshelf, 2013.

[23] M. J. Wise, "YAP3: improved detection of similarities in computer program and other texts," in *27th SIGCSE Technical Symposium on Computer Science Education*, 1996, pp. 130–134.

[24] W. B. Croft, D. Metzler, and T. Strohman, *Search Engines: Information Retrieval in Practice*. Addison-Wesley, 2010.

Work-in-Progress: Syntactic Code Similarity Detection in Strongly Directed Assessments

ORIGINALITY REPORT

| 5% | 2% | 3% | 1% |
|---|---|---|---|
| SIMILARITY INDEX | INTERNET SOURCES | PUBLICATIONS | STUDENT PAPERS |

PRIMARY SOURCES

| | | |
|---|---|---|
| 1 | Oscar Karnalim, Simon. "Common Code Segment Selection: Semi-Automated Approach and Evaluation", Proceedings of the 52nd ACM Technical Symposium on Computer Science Education, 2021<br>Publication | 1% |
| 2 | www.nejm.org<br>Internet Source | <1% |
| 3 | banglajol.info<br>Internet Source | <1% |
| 4 | Submitted to Universitas Brawijaya<br>Student Paper | <1% |
| 5 | Submitted to University of Central Florida<br>Student Paper | <1% |
| 6 | hdl.handle.net<br>Internet Source | <1% |
| 7 | repository.maranatha.edu<br>Internet Source | <1% |

| | | |
|---|---|---|
| 8 | Maresha Caroline Wijanto, Oscar Karnalim, Mewati Ayub, Hapnes Toba, Robby Tan. "Transitioning from Offline to Online Learning: Issues from Computing Student Perspective", 2021 IEEE Global Engineering Education Conference (EDUCON), 2021<br>Publication | <1 % |
| 9 | Oscar Karnalim, Simon. "Disguising Code to Help Students Understand Code Similarity", Koli Calling '20: Proceedings of the 20th Koli Calling International Conference on Computing Education Research, 2020<br>Publication | <1 % |
| 10 | Ricardo Franclinton, Oscar Karnalim, Mewati Ayub. "A Scalable Code Similarity Detection with Online Architecture and Focused Comparison for Maintaining Academic Integrity in Programming", International Journal of Online and Biomedical Engineering (iJOE), 2020<br>Publication | <1 % |
| 11 | www.hindawi.com<br>Internet Source | <1 % |
| 12 | Oscar Karnalim, Simon, William Chivers. "Preprocessing for Source Code Similarity Detection in Introductory Programming", Koli Calling '20: Proceedings of the 20th Koli | <1 % |

Calling International Conference on Computing Education Research, 2020

Publication

|  |  |  |  |
|---|---|---|---|
| Exclude quotes | Off | Exclude matches | Off |
| Exclude bibliography | On |  |  |