Informatics in Education, 2024, Vol. 23, No. 3, 625–645
© 2024 *Vilnius University*
DOI: 10.15388/infedu.2024.21

Simulating Similarities to Maintain Academic Integrity in Programming

Oscar KARNALIM
Maranatha Christian University, Surya Sumantri Street No. 65, Indonesia
e-mail: oscar.karnalim@it.maranatha.edu

Received: August 2023

Abstract. Programming students need to be informed about plagiarism and collusion. Hence, we developed an assessment submission system to remind students about the matter. Each submission will be compared to others and any similarities that do not seem a result of coincidence will be reported along with their possible reasons. The system also employs gamification to promote early and unique submissions. Nevertheless, the system might put unnecessary pressure as coincidental similarities can still be reported. Further, it does not specifically cover self-plagiarism. We revisit the system and shift our focus to report simulated similarities from student own submission instead of reporting actual similarities across submissions. According to our evaluation with 390 students and five quasi-experiments, students with simulated similarities are slightly more aware of plagiarism and collusion, self-plagiarism in particular. Their awareness of the matter is somewhat acceptable (around 75%) and they see the benefits of our assessment submission system.

Keywords: code similarity, simulation, plagiarism, collusion, programming.

1. Introduction

In software development industry, code reuse is common for time-efficiency (Haefliger *et al.*, 2008) as not all parts of the software are written from scratch. Consequently, such a reuse is promoted in academia. However, a number of limitations are imposed to ensure that the reuse still supports student learning process (Zander *et al.*, 2019).

Any reuse instances should be acknowledged (Pangestu and Simon, 2021) so that it is not perceived as a breach of academic integrity in programming (Simon *et al.*, 2013). If the reused code is stolen from the original author(s), the breach is called plagiarism (Fraser, 2014). Otherwise, it is collusion.

To mitigate the incidence of plagiarism and collusion, instructors need to educate students about the matter and penalize those who breach academic integrity (Karnalim *et al.*, 2019). The former is usually conducted manually at the beginning of the course or before issuing an assessment. The latter is conducted after the due date of assessments at which all submissions are checked for originality. An automated similarity detector is typically employed and suspicious submissions are manually investigated.

We previously developed an assessment submission system (referred as INIT-SYS) (Karnalim *et al.*, 2023) that can help instructors in educating students about plagiarism and collusion. Each time a student submits their work, it will be compared with other submissions and any superficial similarities will be reported along with their possible reasons. INIT-SYS employs gamification to further promote student engagement. Students with INIT-SYS have better awareness of plagiarism and collusion. Further, they are less likely to engage in such misconducts.

Despite the benefits, the current system has two limitations. First, as the system is fully automated and long similarities can still be coincidental, the reported similarities are sometimes not evident for raising suspicion. This might put unnecessary pressure to students who are not involved in plagiarism and collusion, affecting their awareness of the matter. It can also reduce the credibility of the system in reporting similarities though students have already been informed that the detection is intentionally designed to be less accurate (so that they cannot learn to trick common similarity detectors for identifying plagiarism and collusion). Second, as the system only report similarities across submissions in a particular assessment, our previous studies show that students have low awareness of self-plagiarism.

In response to the aforementioned gaps, we revisit INIT-SYS and shift our focus from reporting actual similarities to simulated similarities (referred as SIMU-SYS). For each submitted work, a simulation about superficial code similarities will be generated by disguising some parts of the code. In such a manner, no pressure is given to students who are not involved in plagiarism and collusion. Further, the credibility of the system is not harmed as it focuses on simulated similarities. In addition, students are expected to have higher awareness of self-plagiarism since the simulation is based on their own code.

Our study has three research questions:

- **RQ1:** Are students with SIMU-SYS more aware of plagiarism and collusion, self-plagiarism in particular than students with INIT-SYS?
- **RQ2:** How aware are students with SIMU-SYS about plagiarism and collusion? Is it affected by their programming skill?
- **RQ3:** What are student perspective about the benefits of SIMU-SYS?

RQ1 aims to highlight any similarities and differences of SIMU-SYS compared to its predecessor (INIT-SYS). It measures whether students with simulated similarities have better awareness of plagiarism and collusion than those with actual similarities. The discussion covers not only general awareness but also awareness of self-plagiarism, which was low for students with actual similarities. RQ2 aims to capture overall awareness of plagiarism and collusion mainly resulted from the use of SIMU-SYS, while considering programming skill. It specifically reports which awareness aspects of plagiarism and collusion that are high and low. The former is useful to know the strengths of SIMU-SYS while the latter can be a reference for further improvement. RQ3 provides supportive evidences about the benefits of SIMU-SYS via a questionnaire survey. It summarise student confirmation about such benefits.

2. Related Work

Academic integrity refers to actions that promote honesty, trust, fairness, respect, responsibility, and courage (ICAI, 2018). Honesty can be demonstrated by being truthful and giving appropriate credits. Trust can be demonstrated by clearly stating expectation and being transparent in any processes. Fairness can be demonstrated by being consistent and objective. Respect can be demonstrated by showing empathy and promoting active feedback. Responsibility can be demonstrated by being accountable for own actions. Courage can be demonstrated by dealing with discomfort for something right.

Plagiarism and collusion are common breaches of academic integrity in programming. They typically happen due to opportunities, student pressure, and student misrationalization (Albluwi, 2019). Students can be tempted to do plagiarism or collusion if there are opportunities to do so. Such opportunities can be reduced by personalizing assessments (Bradley, 2020), varying assessments across course offerings (Simon, 2017), or generating different assessments for each student (Spinellis *et al.*, 2007). It is also possible to introduce post authentication for student works. Students can be required to present their work (Halak and El-Hajjar, 2016) or to have one-on-one interview with the instructor (Grunwald *et al.*, 2015).

To help instructors identifying plagiarism and collusion cases, an automated similarity detector can be employed. Adkins and Joyner (2022) presented a comprehensive workflow about the use, especially to deal with large classes. Programming similarity detectors are similar to those of text except that they focus more on comparing student submissions to one another rather than comparing those to sources from the internet (Folt`ynek *et al.*, 2019; Lee *et al.*, 2023). The programming similarity detectors can employ conventional matching algorithms (Karnalim *et al.*, 2022a), information retrieval algorithms (Ullah *et al.*, 2021), clustering algorithms (Cheers and Lin, 2023; Ďuračík *et al.*, 2020), or classification algorithms (Hosam *et al.*, 2022). MOSS (Schleimer *et al.*, 2003) and JPlag (Prechelt *et al.*, 2002) are two common examples of publicly available similarity detectors.

It is worth noting that Artificial Intelligence (AI) disrupts programming education (Prather *et al.*, 2023). It can provide more opportunities to cheat (or at least to disguise the plagiarism or the collusion act) (Orenstrakh *et al.*, 2023). Students can use Large Language Models (LLMs) like Github Copilot (Dakhel *et al.*, 2023) or ChatGPT (Kocoń *et al.*, 2023) to inappropriately help them completing assessments. They can also employ code obfuscation tools (Huang *et al.*, 2023) to disguise similarities between the copied work and its original. A number of AI assistance detectors have been developed but their effectiveness is satisfactory (Orenstrakh *et al.*, 2023). Further improvements are needed.

Students can be stressed due to pressure, tempting them to do plagiarism or collusion (Hellas *et al.*, 2017). Pressure regarding task difficulty can be minimized by breaking down large assessments to many smaller assessments (Allen *et al.*, 2018). Time pressure can be addressed by promoting early submissions via additional incentives (Spacco *et al.*, 2013).

Students can be involved in plagiarism or collusion if the act is justified with wrong rationales (e.g., reusing code from previous assessments without acknowledgment is acceptable as many students do it). Student misrationalization is generally addressed by informing students about academic integrity (Simon *et al.*, 2018). Students need to be explicitly aware about which acts constitute of plagiarism or collusion. This can be part of the curriculum or the course syllabus (Greening *et al.*, 2004). However, in practice, such information is briefly delivered either at the beginning of the course or right after issuing an assessment (Simon *et al.*, 2018).

A number of tools have been developed to remind students about academic integrity and the futility of the breaches. Tsang *et al.* (2018) introduced a mobile application that contains modules about ethics. It is also featured with a number of quizzes to test student awareness of the matter.

Le *et al.* (2013) showed the futility of code disguises by allowing students to access MOSS and JPlag similarity reports before final submission. This can inform students that many superficial and syntactical changes will not fool common similarity detectors, discouraging them to be involved in plagiarism and collusion.

Karnalim and Simon (2020) also presented a tool for showing the futility of code disguises. The tool accepts a code file and then disguises some parts of it. Unlike common code obfuscation tools, it ensures that the disguised parts are still readable. Since the tool can be independently used, the disguises are limited to superficial variations.

We developed an assessment submission system that reports relatively long similarities and their possible reasons per submission (referred as INIT-SYS) (Karnalim *et al.*, 2023). Short similarities are likely to be a result of coincidence, not plagiarism and collusion (Mann and Frew, 2006). They might be a result of compilation requirement, legitimately copied code, intuitive implementation, and suggested implementation (Simon *et al.*, 2020). For the purpose of privacy, all information about other students would be anonymized. If no similarities could be reported, a simulation would be generated with comparable information.

To promote student engagement, gamification was employed. Students earned game points and badges by submitting work as unique and as early as possible. Uniqueness aimed to discourage the incidence of plagiarism and collusion while earliness aimed to reduce time pressure, a reason to plagiarize or collude. Ten students with the highest game points would be shown in the leaderboard and non-game incentives (bonus marks or e-money) would be provided for those in top five.

According to our evaluation, students with INIT-SYS were more aware of programming plagiarism and collusion than those with the conventional approach (i.e., manually informed by the instructors). Further, they were less likely to engage in plagiarism and collusion as their submitted programs had lower similarity and fewer cases of plagiarism and collusion were identified. In addition, they completed assessments earlier and engaged more with INIT-SYS.

3. Method

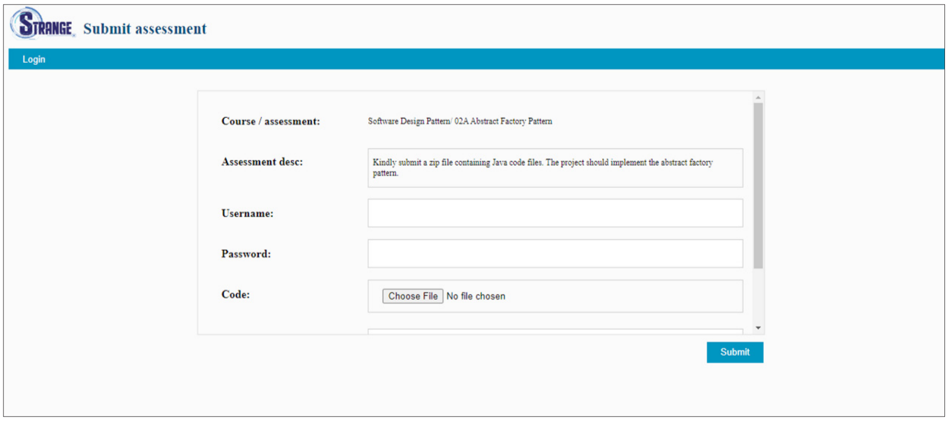
This section discusses about SIMU-SYS, our modified assessment submission system and how to address the three research questions.

3.1. *SIMU-SYS, The Assessment Submission System*

We further developed our assessment submission system (INIT-SYS) so that it reports simulated similarities instead of the actual ones (SIMU-SYS). This is expected to increase student awareness of plagiarism and collusion since no pressure is given to students who are not involved in plagiarism and collusion. It cam also promote awareness of self-plagiarism as simulated similarities are entirely generated from student own submissions.

SIMU-SYS accepts Java or Python submissions at which each submission can be a regular code file or a zip file with multiple code files. For the latter, all contained files will be concatenated prior processed. Fig. 1 shows the submission page and it is assigned with a public link generated for each assessment. If the student has not logged in, they need to enter username and password along with their work. Otherwise, they can simply upload the work.

Each time a student submits their work, a number of code segments will be selected from the submission in three steps. First, the submission is converted to code tokens with the help of ANTLR (Parr, 2013). Second, code segments are formed by taking any identifier tokens that are at the start of code line, and merge all of their following tokens till the length is no less than eight program statements and the last token is at the end of code line. This way, the reported code segments do not look like coincidental matches. Although students are informed that the reported similarities are part of simulation, it is still important to ensure that the report is somewhat evident. Eight program statements is defined as the minimum size threshold based on our manual observation on previous



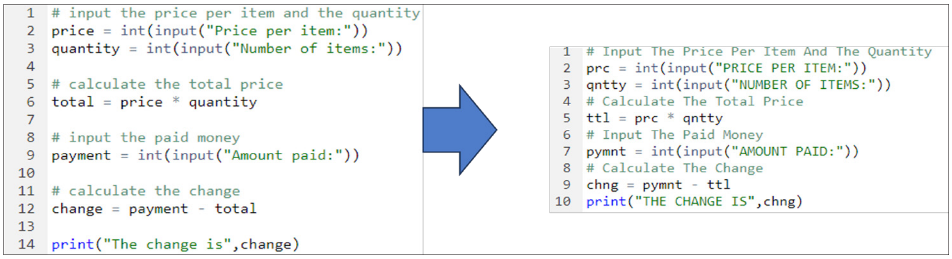
Course / assessment:	Software Design Pattern/ 02A Abstract Factory Pattern
Assessment desc:	Kindly submit a zip file containing Java code files. The project should implement the abstract factory pattern.
Username:	
Password:	
Code:	Choose File No file chosen

Fig. 1. Submission page of the system.

experiments (Karnalim *et al.*, 2023). Third, up to three formed segments are then randomly selected. We limit the formed segments only to those that occur in less than half of the submissions of all students. Otherwise, they are likely to be a result of legitimately copiable code (e.g., code for libraries or base structure) or common ways to solve a task (e.g., code for insertion sort in a task about implementing the sort).

The selected code segments are then disguised. However, the applied disguises are limited to superficial variations nullified by common similarity detectors: comments, white space, identifier names, constants, and some primitive data types (exclusive to Java). Knowledge about such disguises are harmless for students. In total, there are 54 disguises: 24 are about comments, six are about white space, nine are about identifier names, two are about data types, and 13 are about constants. Full list of the disguises can be seen in our previous work (Karnalim *et al.*, 2022b). Fig. 2 shows an example of the code disguises. The left side is the original code while the right side is the disguised one. Both are semantically the same and they only differ in the capitalization, the blank newlines, and the identifier names.

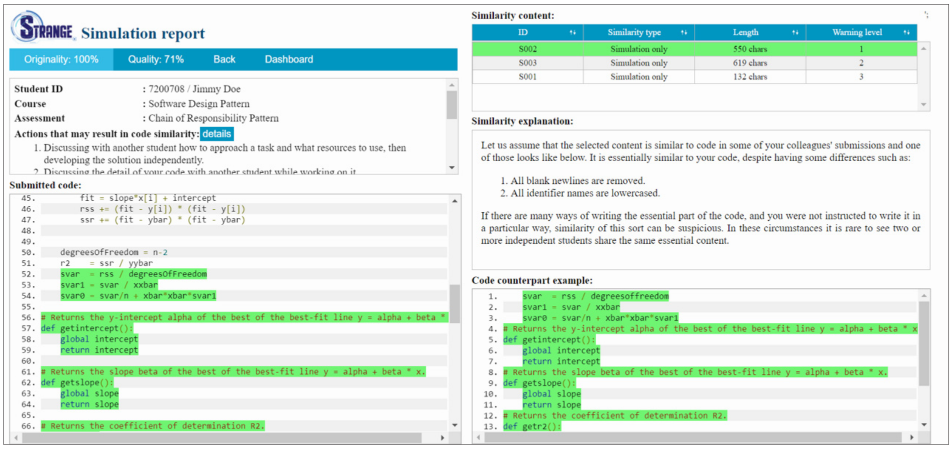
A report containing the submission, its disguised version, and information about programming plagiarism and collusion will be generated and its public unique link will be sent to the submitter via email. An example of the report can be seen in Fig. 3. A menu



```
# input the price per item and the quantity
price = int(input("Price per item:"))
quantity = int(input("Number of items:"))

# calculate the total price
total = price * quantity

# input the paid money
payment = int(input("Amount paid:"))

# calculate the change
change = payment - total

print("The change is",change)
```

```
# Input The Price Per Item And The Quantity
prc = int(input("PRICE PER ITEM:"))
qntty = int(input("NUMBER OF ITEMS:"))
# Calculate The Total Price
ttl = prc * qntty
# Input The Paid Money
pymnt = int(input("AMOUNT PAID:"))
# Calculate The Change
chng = pymnt - ttl
print("THE CHANGE IS",chng)
```

Fig. 2. Example of code disguises focusing on superficial variations.



Student ID : 7200708 / Jimmy Doe
Course : Software Design Pattern
Assessment : Chain of Responsibility Pattern
Actions that may result in code similarity: details

1. Discussing with another student how to approach a task and what resources to use, then developing the solution independently.
2. Discussing the detail of your code with another student while working on it

Submitted code:

```
        fit = slope*x[i] + intercept
        rss += (fit - y[i]) * (fit - y[i])
        ssr += (fit - ybar) * (fit - ybar)


    degreesOfFreedom = n-2
    r2    = ssr / yybar
    svar  = rss / degreesOfFreedom
    svar1 = svar / xxbar
    svar0 = svar/n + xbar*xbar*svar1

# Returns the y-intercept alpha of the best of the best-fit line y = alpha + beta *
def getintercept():
    global intercept
    return intercept

# Returns the slope beta of the best of the best-fit line y = alpha + beta * x.
def getslope():
    global slope
    return slope

# Returns the coefficient of determination R2.
```

Similarity content:

ID	Similarity type	Length	Warning level
S002	Simulation only	550 chars	1
S003	Simulation only	619 chars	2
S001	Simulation only	132 chars	3

Similarity explanation:

Let us assume that the selected content is similar to code in some of your colleagues' submissions and one of those looks like below. It is essentially similar to your code, despite having some differences such as:

1. All blank newlines are removed.
2. All identifier names are lowercased.

If there are many ways of writing the essential part of the code, and you were not instructed to write it in a particular way, similarity of this sort can be suspicious. In these circumstances it is rare to see two or more independent students share the same essential content.

Code counterpart example:

```
    svar  = rss / degreesoffreedom
    svar1 = svar / xxbar
    svar0 = svar/n + xbar*xbar*svar1
# Returns the y-intercept alpha of the best of the best-fit line y = alpha + beta * x
def getintercept():
    global intercept
    return intercept
# Returns the slope beta of the best of the best-fit line y = alpha + beta * x.
def getslope():
    global slope
    return slope
# Returns the coefficient of determination R2.
def getr2():
```

Fig. 3. Example of simulation report.

bar is shown on top at which the submitter can jump to the code quality report (will be discussed briefly later), go back to the previous page, or move to the dashboard page. If not logged in, the link to the dashboard page is replaced with a link for login. The top-left panel contains metadata of the submission and general knowledge of plagiarism and collusion. The top-right panel lists the disguised code segments. If one of them is selected, the segment will be highlighted in the submitted code panel, its disguised version will be shown on the code counterpart example, and its explanation of the disguises will be displayed on the similarity explanation panel. The same behavior occurs when a code segment is selected from the submitted code panel.

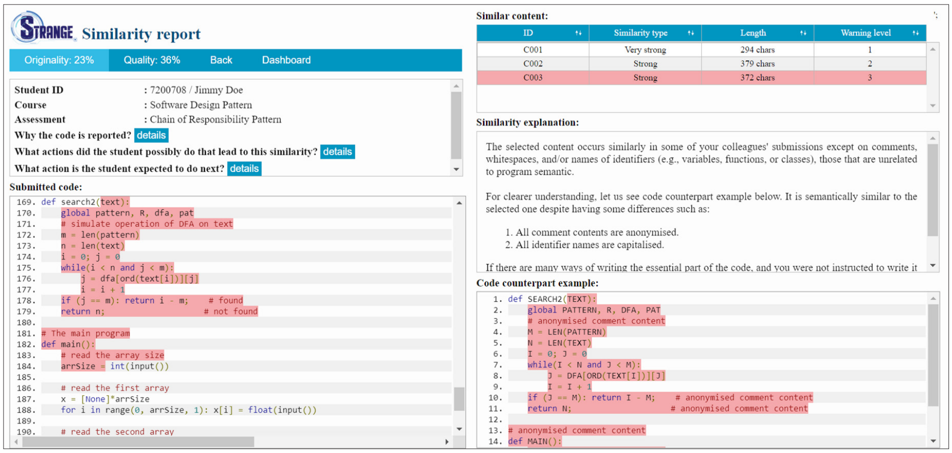
We acknowledge the importance of warning students who are clearly involved in plagiarism and collusion. Hence, if a submission has at least half of its content similar to other submissions and the similarities are not common across all submissions, involved students will be warned by reporting actual similarities.

Before a simulation is generated, the submission is compared to other submissions that have been already submitted for given assessment. The comparison works in four stages. First, the submission is converted to syntax tree, a representation that can capture program structure, with ANTLR. The syntax tree is then linearised to token strings by concatenating *N* tokens in a preorder manner. *N* represents approximate number of tokens for eight program statements; it is defined as 80 for Java and 40 for Python. Second, the token strings are compared to those of other submissions and they will be marked as matches if found. Third, any matches that can be found in at least half of all submissions are ignored as they are likely to be coincidental. Fourth, for each submission, any remaining adjacent matches are merged and if all matches take more than half of that submission, a similarity report will be generated.

The similarity report will be generated in the same manner as the simulation report except that the selected code segments are actual similarities. Further, the disguises are derived from variations between the selected code segments and their match counterparts. For example, if a code segment is considered as a match with that of another submission and its variations are comments and white space, then the disguises can be about removing all comments and adding more blank lines.

The similarity report contains comparable information to that of simulation report. However, the disguised contents are replaced with actual similarities from other submissions. An example of the similarity report can be seen in Fig. 4. General metadata and justifications of reporting can be seen on the top-left panel. The submitted code can be seen on the bottom-left panel. If a code segment is selected, the segment will be highlighted in red instead of green (as in the simulation report). This informs the student that the reported similarities are real, not just a simulation. Selecting a code segment will also trigger the system to show the segment's disguised version (on the code counterpart example panel) and its explanation (on the similarity explanation panel). A table listing all reported code segments is provided on the top-right panel. The menu bar is similar to that of the simulation report.

We allow resubmission by default. However, instructors can still identify students attempting to evade detection of plagiarism and collusion since they have access to all submissions.



Student ID	: 7200708 / Jimmy Doe
Course	: Software Design Pattern
Assessment	: Chain of Responsibility Pattern

Why the code is reported? details
What actions did the student possibly do that lead to this similarity? details
What action is the student expected to do next? details

Submitted code:

```
def search2(text):
    global pattern, R, dfa, pat
    # simulate operation of DFA on text
    m = len(pattern)
    n = len(text)
    i = 0; j = 0
    while(i < n and j < m):
        j = dfa[ord(text[i])][j]
        i = i + 1
    if (j == m): return i - m;    # found
    return n;                      # not found

# The main program
def main():
    # read the array size
    arrSize = int(input())

    # read the first array
    x = [None]*arrSize
    for i in range(0, arrSize, 1): x[i] = float(input())

    # read the second array
```

Similar content:

ID	Similarity type	Length	Warning level
C001	Very strong	294 chars	1
C002	Strong	379 chars	2
C003	Strong	372 chars	3

Similarity explanation:

The selected content occurs similarly in some of your colleagues' submissions except on comments, whitespaces, and/or names of identifiers (e.g., variables, functions, or classes), those that are unrelated to program semantic.

For clearer understanding, let us see code counterpart example below. It is semantically similar to the selected one despite having some differences such as:

1. All comment contents are anonymised.
2. All identifier names are capitalised.

If there are many ways of writing the essential part of the code, and you were not instructed to write it

Code counterpart example:

```
def SEARCH2(TEXT):
    global PATTERN, R, DFA, PAT
    # anonymised comment content
    M = LEN(PATTERN)
    N = LEN(TEXT)
    I = 0; J = 0
    while(I < N and J < M):
        J = DFA[ORD(TEXT[I])][J]
        I = I + 1
    if (J == M): return I - M;    # anonymised comment content
    return N;                     # anonymised comment content

# anonymised comment content
def MAIN():
```

Fig. 4. Example of similarity report.

To further promote student engagement, both SIMU-SYS and INIT-SYS can be featured with gamification. Per assessment, each student gets game points that are equivalent to the percentage of uniqueness times 100 plus number of hours before the deadline. If a student has more than one submission, the points will be averaged. Top-10 students with the highest game points will be shown in a leaderboard.

Students can also obtain badges by fulfilling certain conditions. There are a total of 19 badges. Twelve of them are based on self-progress like submitting a work or opening simulation reports. Six badges are based on competition like submitting the most unique submissions. The remaining one is based on group progress; participating in reaching expected cumulative uniqueness points for an assessment. Each badge equals to 100 game points.

SIMU-SYS and INIT-SYS are not intended to help students trick the detection of plagiarism and collusion with common detectors such as MOSS and JPlag. All reported variations are limited to superficial level and they are commonly nullified by such detectors. Further, as our system immediately reports the similarities, the detection is less accurate; it does not rely on a complete set of submissions. In addition, no information about other involved students are reported.

It is worth noting that SIMU-SYS and and INIT-SYS aim to remind students about plagiarism and collusion. It does not replace the role of instructors maintaining academic integrity in their courses. Instructors still need to inform students about the matter manually at the beginning of the course or prior issuing an assessment. They are also expected to penalize students who are involved in plagiarism and collusion according to the course policy. All submissions need to be compared one another with a common detection tool after the due date and any suspicious submissions should be manually checked for the possibility of misconducts.

Since SIMU-SYS was derived from INIT-SYS, they both essentially have similar features. In addition to shifting from reporting actual similarities to the simulated ones, SIMU-SYS also has several exclusive features. First, instructors and students can be coinstructors for other courses. This facilitates team teaching and involvement of tutors. Second, assessments can be set to accept late submissions. This might be needed for assessments with lenient deadline. Third, for each assessment, the system lists students who have not submitted the work. Instructors might use that information to remind the students. Fourth, a code quality report with static analysis will be provided for each submission. The analysis covers 32 Java code quality issues and 20 Python code quality issues. Further details can be seen on its corresponding publication (Karnalim *et al.*, 2022b).

In terms of aspects of academic integrity, SIMU-SYS can promote trust (being transparent about expectation of plagiarism and collusion), honesty (stressing the need to put appropriate credit), and responsibility (being accountable for own plagiarism acts).

3.2. *Experimental Setup*

Six programming courses employed SIMU-SYS in 2022 and its predecessors (INIT-SYS with or without gamification) in 2021. The details can be seen in Table 1. The courses were from either information technology (IT) or information system (IS) major. IT major is more focused to algorithms and programming while IS major is focused on system and business analysis. They are both enrolled by undergraduates and are expected to be completed in four years (including thesis). Programming level 1 refers to the first programming course in the major; programming level 2 refers to the second programming course; programming level advanced refers to a programming course offered near the end of the study (third year). While level 1 can be referred as CS1, level 2 cannot be referred as CS2 (Hertz, 2010) since for IS major, it is object oriented programming instead of data structure.

For IT major, introductory programming is the first programming course for students. It covers variables, branching, looping, functions, array, matrix, searching, and sorting in Python. One lab and one homework assessments were given weekly; the former should be completed in a two-hour lab sessions while the latter could be completed at home in given week. The course was enrolled by 55 students in 2022 and 45 students

Table 1

Involved courses with their students and responses; 2022 offerings employ SIMU-SYS while 2021 offerings employ its predecessor (INIT-SYS with or without gamification)

Course	Prog. level	2022		2021	
		Students	Responses	Students	Responses
IT introductory programming	1	55	45	45	43
IT data structure	2	34	29	33	31
IT machine intelligence	Adv	33	28	NA	NA
IS introductory programming	1	38	25	35	30
IS object oriented programming	2	42	34	37	31
IS business application prog.	Adv	19	11	19	15

in 2021. The former employed SIMU-SYS while the latter employed INIT-SYS but without gamification.

Data structure is the second IT programming course and it covers linear data structures, linked list in particular. Per week, students were expected to complete one lab and one homework assessments covering the same task: implementing a data structure in Python. Homework assessments were intended to encourage students completing their work that could not be completed in a two-hour lab session. The 2022 batch has 34 students enrolled while the 2021 batch only has 33 students. SIMU-SYS was used in 2022 while INIT-SYS was used in 2021.

Machine intelligence is an advanced programming course for IT undergraduates, typically offered to third-year students. The assessments were designed similarly to those of data structure except that the topics were about implementing artificial intelligence with some Python libraries. SIMU-SYS was employed to the 2022 batch with 33 students. None of its predecessors were employed in the 2021 batch and thus such a batch is not included in our analysis.

Introductory programming is also the first programming course for IS major. The assessment design was the same with that of IT introductory programming. However, searching and sorting were excluded, and the solutions should be written in both Java and Python. Thirty-eight students enrolled to the 2022 batch with SIMU-SYS while thirty-three students enrolled to the 2021 batch with INIT-SYS.

Object oriented programming is the second IS programming course, covering object oriented concepts in Java (including classes, inheritance, and polymorphism), graphical user interface (GUI) with Java Swing, and databases with SQL. It employed the same assessment design as IT introductory programming. For GUI-related assessments, both lab and homework assessments covered the same task. It was intended to encourage students completing their work that could not be completed in the lab session. The 2022 batch has 42 students using SIMU-SYS while the 2021 batch has 37 students using INIT-SYS but without gamification.

Business application programming is an advanced IS programming course covering Java application with databases. The assessment design was similar to that of IT data structure. Both 2022 and 2021 batches have 19 students; the former employed SIMU-SYS while the latter employed INIT-SYS but without gamification.

For all offerings, students were informed about academic integrity at the beginning of the course. Sometimes, instructors reminded them about the matter while completing assessments. Either SIMU-SYS or INIT-SYS was used for all programming assessments.

Submissions were checked for plagiarism and collusion with the help of an automated similarity detector. Any submissions with high similarity were reported and then manually investigated by the instructor(s) or the tutor(s). Students who seemed to be involved in plagiarism or collusion would get zero marks for their corresponding assessments and they would be reminded. If such misconducts were repeated, heavier penalties would be applied.

Appropriate ethics approval has been granted for the study. Anonymous collective analysis is permitted. Analysis was performed with the help of Microsoft Excel and data collection was performed with the help of Google Form.

3.3. *Addressing RQ1: Awareness of Plagiarism and Collusion on Simulated Similarities*

RQ1 is about whether students with simulated similarities are more aware of plagiarism and collusion, self-plagiarism in particular than students with actual similarities. This was addressed via quasi-experiments comparing students with simulated similarities to those with actual similarities. Five 2022 course offerings (which were with simulated similarities, SIMU-SYS) were paired with their corresponding 2021 offerings (which were with actual similarities). Both 2022 and 2021 offerings have comparable number of students (see Table 1) and comparable assessment designs. IT machine intelligence was excluded since its 2021 offering employed neither SIMU-SYS nor INIT-SYS.

At the end of each course offering, students were invited to complete a voluntary survey about plagiarism and collusion. It consisted of 11 scenarios at which students needed to determine whether the scenarios were academically acceptable. They could choose 'do not know' if they were uncertain about the response. The scenarios can be seen in Table 2. These questions mainly cover honesty and responsibility aspects of academic integrity while some of them implicitly cover trust. S01 and S02 are about contract cheating, purchasing code or paying someone to do the work. S03 is about self-plagiarism, reusing own work without sufficient acknowledgment. S04 is about general plagiarism. S05 is about disguising copied work. S06 is about partial copying. S07 and S08 are about discussion for completing the work. S09 and S10 are about asking help for troublesome code. S11 is about replicating features. Correct responses were derived from similar survey used in our previous study (Karnalim *et al.*, 2023) and discussed among instructors of 2022 and 2021 course offerings. S01–S06 and S10 are not academically acceptable for individual assessments while the rest are.

Table 2
Survey scenarios about programming plagiarism and collusion (Karnalim *et al.*, 2023)

ID	Scenario
S01	Purchasing code written by other students to incorporate into your own work
S02	Paying another student to write the code and submitting it as your own work
S03	Basing an assessment largely on work that you wrote and submitted for a previous course, without acknowledging this
S04	Incorporating the work of another student without their permission
S05	Copying another student's code and changing it so that it looks quite different
S06	Copying an early draft of another student's work and developing it into your own
S07	Discussing with another student how to approach a task and what resources to use, then developing the solution independently
S08	Discussing the detail of your code with another student while working on it
S09	Showing troublesome code to another student and asking them for advice on how to fix it
S10	Asking another student to take troublesome code and get it working
S11	After completing an assessment, adding features that you noticed when looking at another student's work

Number of responses per course offering can be seen in Table 1. The proportion to total students is relatively high. The responses were then analyzed in two steps. First, the average percentage of correct responses for both 2022 and 2021 offerings were calculated as a whole set of questions and individually. Second, for each course, its responses from 2022 offering was compared to those of 2021 offering; all significant differences (validated with an unpaired two-tailed t-test with 95% confidence rate) were reported and discussed.

For IT data structure and IS introductory programming, the 2021 offerings employed INIT-SYS. Hence, any differences can be treated as a result of showing simulated similarities instead of the actual ones. However, for IT introductory programming, IS object oriented programming, and IS business application programming, the 2021 offerings employed INIT-SYS without gamification. The differences can be a result of introducing gamification in addition to showing simulated similarities. On INIT-SYS experiments, students with gamification had slightly higher awareness of plagiarism and collusion especially S01 than students without gamification (Karnalim *et al.*, 2023). Hence, any improvements on overall percentage of correct responses and S01 would not be entirely linked to the impact of showing simulated similarities in the analysis.

While it is interesting to measure student awareness of plagiarism and collusion based on the number of identified cases, it was not possible as 2022 offerings were delivered in a hybrid manner (both online and onsite) while 2021 offerings were delivered fully online. Onsite sessions were more supervised and instructors could consider more information such as student behavior and interactions for identifying cases of plagiarism and collusion.

Being more aware of plagiarism and collusion, students are expected to promote at least three aspects of academic integrity: trust, honesty, and responsibility.

This experiment was different to that from our previous study (Karnalim *et al.*, 2023). Our current experiment was more focused on measuring the impact of shifting from reporting actual similarities (INIT-SYS) to reporting simulated similarities (SIMU-SYS). Our previous experiment was more focused on measuring the impact of gamification and employing INIT-SYS. Further, our previous experiment relied on data collected one year earlier. Any differences observed from our current experiment were more likely to be a result of shifting from actual to simulated similarities.

3.4. *Addressing RQ2: Overall Degree of Awareness of Plagiarism and Collusion*

RQ2 is about degree of awareness of students with simulated similarities regarding plagiarism and collusion, and whether it is affected by students' programming skill. This was addressed with the same survey for addressing RQ1. However, only percentages of correct responses of 2022 course offerings (SIMU-SYS) were reported. Further, they were grouped based on the course programming level (1, 2, or advanced). Scenarios with the highest percentages of correct responses would be discussed along with scenarios with the lowest percentages of correct responses. To test the effect of programming skill, results for each programming level would be compared to those of its lower level. Any statistical differences were validated with an unpaired two-tailed t-test with 95% confidence rate.

This experiment was exclusive to our current study. Similar to RQ1, students with high awareness of plagiarism and collusion might be able to promote trust, honesty, and responsibility.

3.5. *Addressing RQ3: Benefits of Our Assessment Submission System that Reports Simulated Similarities?*

RQ3 is about student perspective regarding the benefits of our assessment submission system that reports simulated similarities (SIMU-SYS). This was addressed by asking six additional survey questions (see Table 3) along with the RQ1 survey for 2022 course offerings that employed SIMU-SYS. For S12-S16, students needed to show their agreement toward a particular statement in a 5-point Likert scale (1 for ‘strongly disagree’, 2 for ‘disagree’, 3 for ‘neutral’, 4 for ‘agree’, and 5 for ‘strongly agree’). The results would be analyzed based on the average score of the Likert scale. Any phrases of ‘assessment submission system‘ referred to SIMU-SYS, the system they were using.

S12 and S13 were about the goals of SIMU-SYS: helping students to understand plagiarism and collusion; and discouraging them to be involved in such misconducts. S14 asked about how reasonable the reported similarities, whether they were justifiable. S15 asked about how helpful information given in the simulation reports. S16 was essentially the same to S15 except that it was for the similarity reports. S17 was the only question where students could choose one or more responses. It asked about factors affecting coincidental similarities. The options were derived from a study about common code (Simon *et al.*, 2020): compilation requirement, legitimately copied code, intuitive implementation, suggested implementation, trivial tasks, and strongly directed assessment specifications. An ‘other’ option was also provided so that students could list their own reasons if any. The responses would be analyzed based on occurrence frequencies.

This experiment was exclusive to our current study. By explicitly acknowledging the benefits of SIMU-SYS, students are expected to have values in trust and honesty.

Table 3
Survey questions about student perspectives

ID	Statement
S12	The assessment submission system helps me to understand plagiarism and collusion
S13	The assessment submission system discourages me to be involved in plagiarism and collusion
S14	Both reported simulated and actual similarities are reasonable for raising suspicion
S15	Information given in simulated similarities are helpful for me to understand plagiarism and collusion
S16	Information given in actual similarities are helpful for me to understand plagiarism and collusion
S17	Based on your observation on the reported similarities, which factors can entail coincidental similarities?

4. Results and Discussion

This section reports our findings from addressing the research questions and discusses them.

4.1. *Awareness of Plagiarism and Collusion on Simulated Similarities*

Fig. 5 shows that students with simulated similarities (SIMU-SYS) had slightly better awareness of plagiarism and collusion to students with actual similarities (INIT-SYS with or without gamification). In IS introductory programming and IS object oriented programming, overall percentage of correct responses was increased in a statistically significant manner by 6% (p = 0.03) and 11% (p = 0.01) respectively. Increase on IS object oriented programming was larger since the differences between both groups also included gamification, which seemed to positively affect overall awareness (Karnalim *et al.*, 2023). The improvement was not substantial since in both SIMU-SYS and INIT-SYS, each student still got a similarity report (regardless whether it was just a simulation or not) at which they could learn about plagiarism and collusion.

Students with simulated similarities had better awareness of self-plagiarism. S03 experienced statistically significant improvement on three quasi-experiments: IT introductory programming (p = 0.01 with 27% improvement), IT data structure (p < 0.01 with 40% improvement), and IS introductory programming (p = 0.01 with 23% improvement). In simulated similarities, students were explicitly informed that the disguised code were from their own submissions. They might realize that code similarities could be a result of reusing own code and such reuse needed acknowledgment.

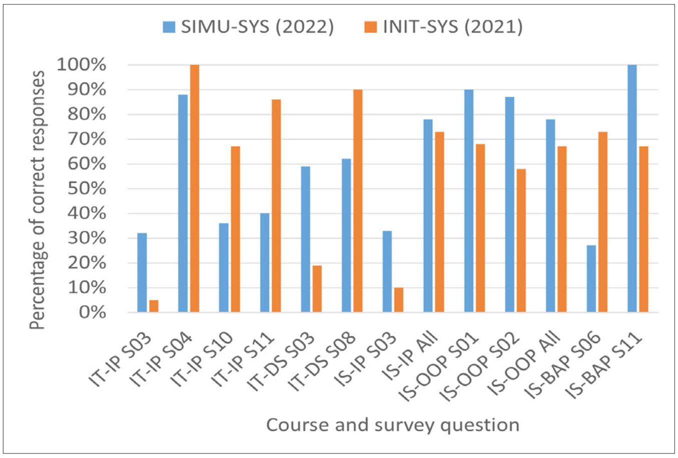


Fig. 5. Statistically significant changes from quasi-experiments; SIMU-SYS vs INIT-SYS; IP refers to introductory programming, DS refers to data structure, OOP refers to object oriented programming, and BAP refers to business application programming.

When other scenarios were individually analyzed, students with simulated similarities occasionally had higher awareness on not purchasing code (S01), not paying another student to complete assessments (S02), and replicating features (S11). S01 improved awareness might be a result of higher engagement introduced by gamification (Karnalim *et al.*, 2023), not replacing actual similarities with the simulated ones.

S02 and S11 were improved since students involved in plagiarism and collusion would feel more warned. SIMU-SYS only generated actual similarities for obvious cases of plagiarism and collusion. Submission written by another student (S02) was likely to be obviously similar to their own. Replicating features (S11) did not always entail the same code, although it was only applicable for complex features. As seen in Table **??**, S11 awareness was increased on IS business application programming (an advanced course) but it was reduced on IT introductory programming.

Other scenarios at which students with simulated similarities occasionally had lower awareness on are not incorporating another student's work without permission (S04), not using another student's draft as the basis of work (S06), and not asking another student to fix the code (S10). SIMU-SYS seldom reported S04, S06 and S10 as actual similarities as they did not entail excessive similarities.

S08 awareness was decreased on IT data structure since assessments of students with simulated similarities were slightly more constrained: class and method names were defined. To avoid suspicion of plagiarism and collusion, students might be more reluctant to discuss the details of their work. The solutions were more likely to be similar although the similarities were common and would not reported.

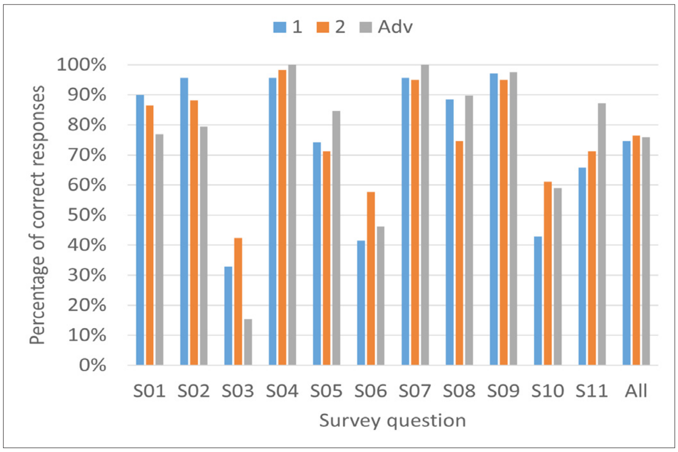


Fig. 6. Student awareness of plagiarism and collusion per programming level (1, 2, or advanced).

4.2. *Overall Degree of Awareness of Plagiarism and Collusion*

Fig. 6 depicts that in general, student awareness of plagiarism and collusion was somewhat acceptable: 74% for level 1 programming courses, 76% for level 2 programming courses, and 76% for advanced level programming courses. They were comparable across programming levels; an unpaired two-tailed t-test with 95% confidence rate showed that the differences were insignificant. Student awareness was not heavily affected by programming skill.

For level 1 programming (IT and IS introductory programming), students were the most aware of S09 (97%), S02 (96%), S04 (96%), and S07 (96%). Students were aware that discussing how to approach a task (S07) and asking for advice (S09) were both allowed as the code was still written independently. They were also aware that incorporating another student's work (S04) was not acceptable though it was seldom reported by SIMUSYS. Some students might be aware about the matter since permission was expected when using other students' work. In IT introductory programming, S04 awareness was reduced when actual similarities were replaced with the simulated ones (see Table **??**. However, the awareness was still relatively high (88%).

Paying another student to complete an assessment (S02) was another scenario at which students were most aware of. The submission could be excessively similar to that student's own and then reported by the system.

Although students had higher awareness of self-plagiarism (S03), the percentage of correct responses was still relatively low (33%). SIMU-SYS should be updated so that it can report historical similarities across assessments and even course offerings. Student awareness of not using another student's draft as the basis of their work (S06) and not asking another student to fix troublesome code (S10) were also relatively low. This was probably because both scenarios seldom resulted in excessive similarities. Another way to inform students about the matter is needed on SIMU-SYS.

As seen in Fig. 6, students in level 2 programming courses had comparable awareness to those of students in level 1 courses. S02, S04, S07, and S09 were scenarios students most aware of while S03, S06, and S10 were scenarios students least aware of.

When observed per scenario, students in level 2 programming courses had lower awareness of discussing the details of the code while working on it (S08) than students in level 1 courses ($p = 0.04$). The assessments introduced structures and templates to follow. Some students might be worried that such a discussion could result in similar solutions, though common code was automatically excluded by SIMU-SYS.

Introduced structures and templates to follow were also the reason why students in level 2 programming courses were more aware of not asking another student to fix the code (S10 with $p = 0.04$). The resulted code was more likely to be similar to the student's.

Students in advanced level programming courses had comparable overall awareness of plagiarism and collusion to those in level 2 programming courses. Statistically significant differences were only observed on S04 ($p < 0.01$) and S08 ($p = 0.04$). Students in advanced level programming had lower awareness of not incorporating the work of another student (S04) than those in level 2 programming. Advanced assessments expected long solutions; incorporating another student's work was less likely to be reported as the proportion of the incorporated work toward the whole submission was typically small.

Students in advanced level programming had higher awareness of discussing the details of the code while working on it (S08). Advanced assessments expected complex solutions with many variations. It was unlikely that such a discussion would result in the same solutions.

4.3. *Benefits of the Assessment Submission System that Reports Simulated Similarities*

Fig. 7 shows that students generally agreed that SIMU-SYS helped them to understand programming plagiarism and collusion (S12 with 4.1 of 5). It prevented them from doing such misconduct (S13 with 4.4 of 5). SIMU-SYS informed students about plagiarism and collusion and that information might discourage them to do plagiarism or collusion. Both goals of SIMU-SYS were acknowledged by the students.

Students benefited from both simulation reports (S15 with 4 of 5) and similarity reports (S16 with 3.9 of 5). These reports provided information about plagiarism and collusion, and some examples of code similarities. The reported similarities looked reasonable for them (S14 with 3.8 of 5). The similarities were unlikely to be a result of coincidence and they were relatively long.

Students were aware that coincidental similarities can be a result of legitimately copied code (71%), suggested implementation (74%), trivial tasks (74%), and strongly directed assessment specifications (72%) However only some of them were aware that compilation requirement (56%) and intuitive implementation (36%) could also entail coincidental similarities. Instructors might want to remind them about the matter in some assessments.

Four 'other' responses were discovered. One of them could be mapped to intuitive implementation while the other three were about using the same code from internet

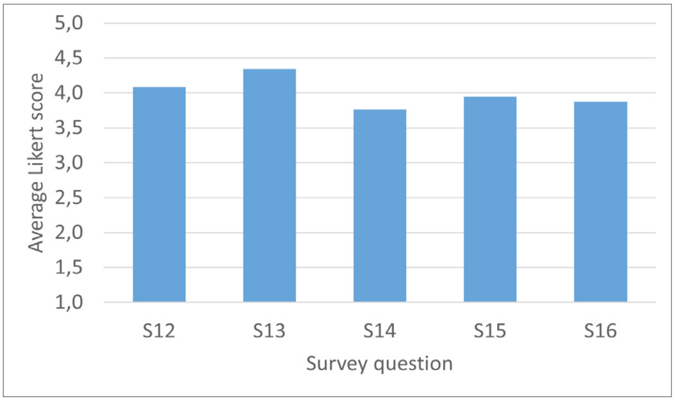


Fig. 7. Student perspective of benefits of SIMU-SYS.

(2%). Search engines might recommend the same site for students while addressing a particular programming issue. Instructors might inform students that this case might happen but it was acceptable so long as the use of the resource was allowed and acknowledged.

4.4. *Discussion*

RQ1 asks whether students with simulated similarities are more aware of plagiarism and collusion, self-plagiarism in particular than students with actual similarities. Our experiments showed that students with simulated similarities are a little bit more aware of plagiarism and collusion. In two of five quasi-experiments, the overall awareness was slightly increased (9% in average). They were also more aware of self-plagiarism as in three quasi-experiments, such awareness was improved by 30% in average.

Students with SIMU-SYS can be more aware of not purchasing code, not paying another student to complete assessments, and replicating features. However, they might be less aware of not incorporating another student's work without permission, not using another student's draft as the basis of work, and not asking another student to fix the code. Each was confirmed by one quasi-experiment.

RQ2 asks how aware students with simulated similarities about plagiarism and collusion and whether it is affected by their programming skill. Our surveys reported that such students are somewhat aware of plagiarism and collusion with around 75% correct responses. The overall degree of awareness was comparable one another across programming levels with no statistical significance was found. Students at a particular programming level might have better awareness on some scenarios than another.

RQ3 asks for supportive evidences about the benefits of SIMU-SYS. Our surveys showed that students generally agreed that SIMU-SYS helped them to understand plagiarism and collusion, discouraging them to do both misconducts. They believed both simulation and similarity reports are helpful. Further, the reported simulated and actual similarities look reasonable. Students were aware that coincidental similarities can occur due to various reasons including legitimately copied code, suggested implementation, trivial tasks, and strongly directed assessment specifications.

5. Limitations

Our study has a number of limitations. First, while students with SIMU-SYS were treated similarly to those with INIT-SYS, we acknowledged that the reported differences might be slightly affected by change in the course delivery mode, from online (2021) to hybrid (2022). Second, though the study was performed on six courses, these courses were from a single institution in a particular country. The findings might not be applicable to other courses and institutions. Third, like many quasi-experiments, we acknowledge that unknown external factors might affect our findings.

6. Conclusions and Future Work

We present SIMU-SYS, an assessment submission system that simulates similarities instead of reporting the actual ones. Our evaluation shows that such a modification slightly increase student awareness of plagiarism and collusion, self-plagiarism in particular. Students are relatively aware of plagiarism and collusion with around 75% correct response rate and it is only slightly affected by programming skill. They also agreed about the benefits of SIMU-SYS and they are aware of some reasons for coincidental similarities.

For future work, we plan to further promote student awareness of plagiarism and collusion by employing small quizzes as part of the gamification. This might encourage students to think further about the matter. We are also interested to replicate the study in other courses and institutions with the same course delivery mode.

Acknowledgements

The research was supported by the Research Institute and Community Service (LPPM) at Maranatha Christian University Indonesia.

References

Adkins, K.L., Joyner, D.A. (2022). Scaling anti-plagiarism efforts to meet the needs of large online computer science classes: Challenges, solutions, and recommendations. *Journal of Computer Assisted Learning*, 38(6), 1603–1619.

Albluwi, I. (2019). Plagiarism in programming assessments: a systematic review. *ACM Transactions on Computing Education*, 20(1), 6–1628.

Allen, J.M., Vahid, F., Downey, K., Edgcomb, A.D. (2018). Weekly programs in a CS1 class: Experiences with auto-graded many-small programs (MSP). In: *ASEE Annual Conference & Exposition*.

Bradley, S. (2020). Creative assessment in programming: Diversity and divergence. In: *Fourth Conference on Computing Education Practice*, pp. 13–1134. 9781450377294.

Cheers, H., Lin, Y. (2023). Identifying plagiarised programming assignments based on source code similarity scores. *Computer Science Education*, 33(4), 621–645.

Dakhel, A.M., Majdinasab, V., Nikanjam, A., Khomh, F., Desmarais, M.C., Jiang, Z.M.J. (2023). Github copilot ai pair programmer: Asset or liability? *Journal of Systems and Software*, 203, 111734.

Ďuračík, M., Hrkút, P., Kršák, E., Toth, S. (2020). Abstract syntax tree based source code antiplagiarism system for large projects set. *IEEE Access*, 8, 175347–175359.

Folt`ynek, T., Meuschke, N., Gipp, B. (2019). Academic plagiarism detection: A systematic literature review. *ACM Computing Surveys (CSUR)*, 52(6), 1–42.

Fraser, R. (2014). Collaboration, collusion and plagiarism in computer science coursework. *Informatics in Education*, 13(2), 179–195.

Greening, T., Kay, J., Kummerfeld, B. (2004). Integrating ethical content into computing curricula. In: *Sixth Australasian Conference on Computing Education*, pp. 91–99.

Grunwald, D., Boese, E., Hoenigman, R., Sayler, A., Stafford, J. (2015). Personalized attention @ scale: talk isn't cheap, but it's effective. In: *46th ACM Technical Symposium on Computer Science Education*, pp. 610–615.

Haefliger, S., von Krogh, G., Spaeth, S. (2008). Code Reuse in Open Source Software. *Management Science*, 54(1), 180–193.

Halak, B., El-Hajjar, M. (2016). Plagiarism detection and prevention techniques in engineering education. In: *11th European Workshop on Microelectronics Education*, pp. 1–3. 978-1-4673-8584-8.

Hellas, A., Leinonen, J., Ihantola, P. (2017). Plagiarism in take-home exams: Help-seeking, collaboration, and systematic cheating. In: *22nd ACM Conference on Innovation and Technology in Computer Science Education*, pp. 238–243.

Hertz, M. (2010). What do "CS1" and "CS2" mean? Investigating differences in the early courses. In: *41st ACM Technical Symposium on Computer Science Education*, pp. 199–203.

Hosam, E., Hadhoud, M., Atiya, A., Fayek, M. (2022). Classification feature sets for source code plagiarism detection in Java. *Journal of Engineering and Applied Science*, 69(1), 1–18.

Huang, W., Meng, G., Lin, C., Yan, Q., Chen, K., Ma, Z. (2023). Are our clone detectors good enough? An empirical study of code effects by obfuscation. *Cybersecurity*, 6(1), 14.

ICAI (2018). The Fundamental Values of Academic Integrity. `https://academicintegrity.org/images/pdfs/20019_ICAI-Fundamental-Values_R12.pdf`.

Karnalim, O., Simon (2020). Disguising code to help students understand code similarity. In: *20th Koli Calling International Conference on Computing Education Research*, pp. 13–1135.

Karnalim, O., Simon, Chivers, W. (2019). Similarity detection techniques for academic source code plagiarism and collusion: a review. In: *International Conference on Engineering, Technology and Education*, pp. 1–6.

Karnalim, O., Simon, Chivers, W. (2022a). Layered similarity detection for programming plagiarism and collusion on weekly assessments. *Computer Applications in Engineering Education*, 30(6), 1739–1752.

Karnalim, O., Simon, Chivers, W. (2023). Gamification to help inform students about programming plagiarism and collusion. *IEEE Transactions on Learning Technologies*, 16(5), 708–721.

Karnalim, O., Simon, Chivers, W., Panca, B.S. (2022b). Automated Reporting of Code Quality Issues in Student Submissions, 517–529. Springer.

Kocoń, J., Cichecki, I., Kaszyca, O., Kochanek, M., Szydło, D., Baran, J., Bielaniewicz, J., Gruza, M., Janz, A., Kanclerz, K., *et al.* (2023). ChatGPT: Jack of all trades, master of none. *Information Fusion*, 101861.

Le, T., Carbone, A., Sheard, J., Schuhmacher, M., De Raath, M., Johnson, C. (2013). Educating computer programming students about plagiarism through use of a code similarity detection tool. In: *2013 International Conference on Learning and Teaching in Computing and Engineering*, pp. 98–105. 9780769549606.

Lee, G., Kim, J., Choi, M.-s., Jang, R.-Y., Lee, R. (2023). Review of code similarity and plagiarism detection research studies. *Applied Sciences*, 13(20), 11358.

Mann, S., Frew, Z. (2006). Similarity and originality in code: Plagiarism and normal variation in student assignments. In: *Eighth Australasian Conference on Computing Education*. ACE 2006, pp. 143–150. 1920682341.

Orenstrakh, M.S., Karnalim, O., Suarez, C.A., Liut, M. (2023). Detecting LLM-Generated Text in Computing Education: A Comparative Study for ChatGPT Cases.

Pangestu, M.A., Simon (2021). Work in progress: An automated management system for references in programming code. In: *IEEE Global Engineering Education Conference*, pp. 1301–1305.

Parr, T. (2013). *The Definitive ANTLR 4 Reference*.

Prather, J., Denny, P., Leinonen, J., Becker, B.A., Albluwi, I., Craig, M., Keuning, H., Kiesler, N., Kohn, T., Luxton-Reilly, A., *et al.* (2023). The robots are here: Navigating the generative ai revolution in computing education. In: *2023 Working Group Reports on Innovation and Technology in Computer Science Education*, pp. 108–159.

Prechelt, L., Malpohl, G., Philippsen, M. (2002). Finding plagiarisms among a set of programs with JPlag. *Journal of Universal Computer Science*, 8(11), 1016–1038.

Schleimer, S., Wilkerson, D.S., Aiken, A. (2003). Winnowing: Local algorithms for document fingerprinting. In: *International Conference on Management of Data*, pp. 76–85. 158113634X.

Simon (2017). Designing programming assignments to reduce the likelihood of cheating. In: *19th Australasian Computing Education Conference*, pp. 42–47.

Simon, Cook, B., Sheard, J., Carbone, A., Johnson, C. (2013). Academic integrity: Differences between computing assessments and essays. In: *13th Koli Calling International Conference on Computing Education Research*, pp. 23–32. 9781450324823.

Simon, Sheard, J., Morgan, M., Petersen, A., Settle, A., Sinclair, J. (2018). Informing students about academic integrity in programming. In: *20th Australasian Computing Education Conference*, pp. 113–122. 9781450363402.

Simon, Karnalim, O., Sheard, J., Dema, I., Karkare, A., Leinonen, J., Liut, M., McCauley, R. (2020). Choosing code segments to exclude from code similarity detection. In: *Working Group Reports on Innovation and Technology in Computer Science Education*, pp. 1–19. 9781450382939.

Spacco, J., Fossati, D., Stamper, J., Rivers, K. (2013). Towards improving programming habits to create better computer science course outcomes. In: *18th ACM Conference on Innovation and Technology in Computer Science Education*, pp. 243–248.

Spinellis, D., Zaharias, P., Vrechopoulos, A. (2007). Coping with plagiarism and grading load: randomized programming assignments and reflective grading. *Computer Applications in Engineering Education*, 15(2), 113–123.

Tsang, H.H., Hanbidge, A.S., Tin, T. (2018). Experiential learning through inter-university collaboration research project in academic integrity. In: *23rd Western Canadian Conference on Computing Education*.

Ullah, F., Jabbar, S., Mostarda, L. (2021). An intelligent decision support system for software plagiarism detection in academia. *International Journal of Intelligent Systems*, 36(6), 2730–2752.

Zander, C., Eckerdal, A., McCartney, R., Mostrom, J.E., Sanders, K., Thomas, L. (2019). Copying can be good: How instructors use imitation in teaching programming. In: *ACM Conference on Innovation and Technology in Computer Science Education*. ACM, pp. 450–456. 9781450368957.

O. Karnalim is an assistant professor at Maranatha Christian University, Indonesia. He is also the head of the Master of Informatics program. His primary research interest is computing education and programming, and he has published nearly a hundred papers on that topic, many of which are indexed in IEEE Xplore, ACM Digital Library, and ScienceDirect. He also serves as an editor of Wiley's Scientific Programming and JISEBI. Further, he is a reviewer on many journals and conferences including IEEE Access, Wiley's CAE, Heliyon, IEEE ToE, IEEE SMC, and IEEE TALE. He is a member of IEEE, IPSJ, IGIP, IAII, and APTIKOM.