A detection process to create awareness of source-code plagiarism among students using it to pass introductory programming[1]

Imelda Smit, North-West University, South Africa
Eduan Naudé, North-West University, South Africa
Busisiwe Zulu, North-West University, South Africa

ABSTRACT

The COVID-19 pandemic restrained the academic environment and changed the rules of the educational game; contact classes were restricted, and online assessments had to be introduced. This situation opened an opportunity for some students to use source-code plagiarism to pass coding assessments in the Introduction to Programming subject module. The focus of this paper is on making sense of this environment to establish a process to ensure that students obtain the skills they need to build on in subsequent modules. This is necessary to reach the outcomes of a computing course. Four aspects were used in establishing this source-code plagiarism awareness process in focusing on one class of students. Qualitative data were gathered by firstly requesting the class to supply feedback on their understanding of source-code plagiarism, and secondly inviting students identified as guilty of Python source-code plagiarism to start a conversation with the lecturer, which was triangulated with quantitative data regarding the success of the latter group in terms of their pass rate. Although the Measure of Software Similarity tool was instrumental in establishing a source-code plagiarism detection process, it is cumbersome and time consuming. Hence fourthly, it was compared to other available tools to determine their suitability in comparison. A refined source-code plagiarism awareness process is the resultant finding of this paper.

Keywords: introductory programming, source-code plagiarism, source-code plagiarism detection tool, source-code plagiarism awareness, source-code plagiarism categories

INTRODUCTION

Academic misconduct is an ongoing issue, but with the COVID-19 pandemic enforcing restrictions in education, it is difficult to ensure that students are working on their own when they do so without supervision and guidance at their place of residence. In a study, 93% of lecturers perceive that students cheat more in an online environment when compared to contact learning (Newton, 2020). In the context of novice programmers learning to code, cheating is more troublesome than in other environments where text plagiarism is an issue, since students do not learn the skill when a simple problem aiming to facilitate the understanding of a construct is copied and pasted. Therefore, the main objective of this paper is to develop a source-code plagiarism detection process which creates awareness among students who make themselves guilty of source-code plagiarism. This is achieved in this study by determining the class's

[1] Date of Submission: 6 March 2023
Date of Review Outcome: 18 December 2023
Date of Acceptance: 1 March 2024

plagiarism awareness, having conversations with students found guilty, and determining whether these students are weakening their chances of passing the module. A secondary objective includes determining whether MOSS should be replaced with another tool in this context. This is important since this step is pivotal in creating awareness, and when it becomes integrated with assessment, student awareness will be ensured. Subsequent sections address the literature applicable to this paper (Section 2), the research methodology used (Section 3), the research design (Section 4), the findings (Section 5), and lastly, concluding the paper (Section 6).

LITERATURE REVIEW

The literature categories of source-code plagiarism (SCP), source-code plagiarism detection (SCPD) tools, Python and its assessment, and extant research on SCP is discussed.

Categories of source-code plagiarism (SCP)

When students start their journey of becoming a programmer, they learn the basic programming concepts; one concept at a time. This involves understanding the problem given and solving it in the context of the programming language; a time-intensive process.

In Table 1, Joy et al. (2010, 2011) categorise SCP. Although categorisation is helpful, not all of the categories are equally applicable to a beginner class of programming. Firstly, in the context of this study, category 3 has the biggest impact; when students struggle to learn the concepts or procrastinate on a potentially difficult task such as writing a program, possibly because they lack the confidence to solve the problem, and then leave it for the last minute with not enough time to solve the problem (Gomes & Mendes, 2007). To complete the assignment in time, such students may revert to copy code from a peer in class – with the solution already available – without learning how to solve the problem (Ngo, 2016). Erkaya (2009) found that lack of motivation, and lack of resources to learn from may be contributing factors.

Table 1:
Six categories of SCP with its impact in the context of this study

Nr.	Category	Explanation	Impact
1	Re-use code as self-plagiarism	Driven by a lack of understanding of how reusing previously submitted coursework might lead to self-plagiarism.	4
2	Using books and other sources	Students using code from a book or another source without giving proper acknowledgements.	2
3	Copying code from peers	Students that copy code from peers, either because they do not know how to solve the problem, or they do not have enough time to complete the coding on their own.	1 (highest)
4	Cooperation with peers	Students working together on an assignment.	3
5	Changing code from one programming language to another	Students find a solution on a program in a different language and then translate it to another language.	4
6	Falsification	Students submit a program or script that does not work the way it should but displays the correct output.	5 (none)

Secondly, students may revert to using books and other sources such as websites (category 2) from which to copy. Zarfsaz and Ahmadi (2017) found that students felt that code obtained from the Internet is well-written and guarantees a good mark, whereas code developed by themselves, would not be as good. This scenario does not provide a readily available solution as in the previous category, and it may be necessary to make changes to the code. Thirdly, cooperating with peers (category 4) is related to category 3, and it is a good way used by students to learn to program (Ngo, 2016). They share ideas and understanding which leads to SCP that students are not aware of, because they are not educated enough on SCP (Maharajh, 2020).

Both re-use of code and using code from another programming language (category 1 and 5) have a limited impact on the students in this context, because of the beginner nature of the course. Falsification (category 6) has no impact in the context of this study, since students only submit .py files, which is run by the assessors to check the output.

Source-code plagiarism detection tools

In the context of a student learning to code, copying code to solve a simple problem defeats the purpose of learning the programming skill. Having said this, it is fair to mention that a student needs to start learning somewhere, and therefore supplied material is seen as *planned plagiarism,* and distinguished from *purposeful plagiarism*, when material outside this realm is copied (Joy et al., 2010).

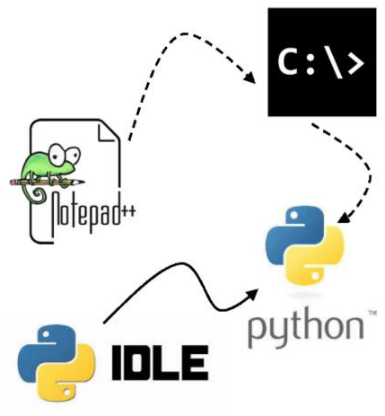
With plagiarism being a long-standing challenge in the academic setting, tools such as Turnitin, Grammarly, and Duplichecker have been used widely in academic writing (Bhosale, 2022). Unfortunately, SCP detection is not done by these tools and a different type of tool is needed to identify copied code. Many SCPD tools are available on the market and examples include: CodeGrade, Codequiry, Measure of Software Similarity (Moss), Unicheck, Plaggie, Sherlock, Marble, and CopyLeaks (Younas, 2021). The SCPD tools use a variety of algorithms to detect SCP in programs. In particular contexts, it may be valuable to have access to tools that can do both text and code plagiarism detection, such as SmallSEOTools.

Although checking the similarity of code is not difficult, finding a culprit is much harder. Many people who copy code will attempt to make some structural changes so that the source-code appears authentic. Therefore, it is very difficult to detect similarity without the help of a proper tool that looks deep into the code (Younas, 2021). The Moss tool was used in 2021 and 2022 at the NWU to identify students guilty of SCP. Since the Moss tool cannot be integrated with the learning management system, can only compare student scripts with one another, is not accessible to students, and is time intensive to use, a list of criteria was compiled to compare SCPD tools – in an attempt to determine how efficient other tools are at detecting SCP in general, and then specifically in a first-year Python assessment context. Tools were then ranked in terms of its effectiveness. It should be emphasized once more that a SCPD tool is not a magic wand, and that a knowledgeable human needs to make a final decision regarding SCP incidents.

Python and its assessment

In the context of this study, Python 3 is the programming language students are using to learn to code. Python is an interpreted language, and one can run simple Python expressions and statements in an interactive programming environment, or the shell. The Integrated DeveLopment Environment (IDLE) is the environment that comes with the Python installation (Lambert, 2018). When launching IDLE, the Python Shell opens to make an editor available – to write a script and access the Python interpreter. In addition, an editor, Notepad++ is made available to students, and they are guided to use the Windows command-line to run Python commands. All the necessary software (Figure 1) is free to download and use.

Figure 1:
Software used



With no automation tool for the assessment of scripts, such as Jype, assessment is a time intensive process, therefore assistants (senior students) are employed to assess the assignments and some tests (Helminen & Malmi, 2010). Assistants responsible for assessments were made aware to look for tell-tale signs giving away SCP. Although a small percentage of students were identified as copying code during the marking of assignments and tests, the large number of students complicated its identification. The fact that online, at-home assessment has been the only option when COVID-19 started, especially in the context of a contact university, made this an unmanageable situation. Students were left to their own devices when studying, although they were in need of a watchful eye - as it were - to keep them learning in a constructive way (Miller, Visser & Staub, 2005).

Source-code plagiarism research

Although source-code plagiarism is only a small part of plagiarism as such, multiple sources indicate that it is a substantial problem, specifically so when it comes to novice students learning to code, and more so during the COVID-19 pandemic. In a study conducted among computing academics in the United Kingdom, Cosma and Joy (2008) determined that *re-using* code, which includes translating code from one programming language to another, as well as utilising code generation software, *obtaining* code, which includes paying another person to write the code, stealing code, and collaborating with peers on individual assignments, and inadequately acknowledging sources by either not listing sources, of fabricating sources, or providing a false reference, constitute SCP. A follow-up study involving students, again in the United Kingdom, Joy et al. (2010) established that SCP is increasingly occurring mostly due to the wealth of online and other resources available to students. In an introductory programming course Karnalim (2017) found that among 400 code pairs suspected of plagiarism, 'modified comment & whitespace' and 'modified identifier names' occurred most frequently, the simplest forms of modifications to be made to copied code. Similar findings were made in a basic programming course, where Maryono, Yuana and Hatta (2019) identified mostly lexical modifications on plagiarised work, which do not require high level programming skills including 'formatting source-code', 'changing comments', 'renaming identifiers', and 'splitting or merging declaration variables'. Lastly, Newton and Essex (2023) posit that the immediacy of the COVID-19 lockdown transition to online assessments challenged the academic environment to put security measures in place, which certainly have increased students' opportunities to make use of SCP to pass programming subjects.

RESEARCH METHODOLOGY

Mainly qualitative data, where the meanings students assigned to their situation, were gathered (Myers, 1997). Firstly, students identified as guilty of Python source-code plagiarism were encouraged to start a conversation with the lecturer. This aspect had the intention to ensure fairness by hearing the student's voice. Secondly, the class was asked to supply feedback on their understanding of source-code plagiarism. The third aspect focused on quantitative data and had the intention of triangulating the success of the group of students guilty of source-code plagiarism in terms of their pass rate with the first two aspects (Thurmond, 2001). Although the Measure of Software Similarity SCPD tool was instrumental in establishing a source-code plagiarism detection process, it had limitations on crucial aspects, and is cumbersome and time consuming to use, and therefore as fourth aspect it was compared to other available tools to determine their suitability in comparison. In this paper, interpretivist content analysis (Erlingsson & Brysiewicz, 2017) is utilised as a qualitative technique to make sense of student feedback and analyse communication with students.

RESEARCH DESIGN

The design of this study involved analysing qualitative feedback received from students who elected to complete an open-ended questionnaire on their experience, the SCP detection process of identifying a sub-set of plagiarism cases where SCP were identified among the class of students and allowing students to state their case by communicating with the lecturer regarding their case(s) - with the purpose of determining the group of students to be penalised for SCP, and tracking the SCP group to determine their quantitative success at the conclusion of the module. Lastly, suitable SCPD tools were investigated using Design Science Research (DSR), a problem-solving approach which increases human understanding through the creation of artefacts or artefact designs. Existing artefacts may be deployed in new ways to facilitate a fresh approach to problems. Results of DSR include new artefacts and design knowledge – to provide better insight (Vom Brocke, Hevner & Maedche, 2020). Due to the extensive process of building cyclic tables of comparison, and it being a lengthy process, only the results are reported on in this paper.

Participants

Across the three North-West University (NWU) campuses and the Open Distance Learning mode, a large number of students from more than 10 courses enrol for the module of which Introduction to Python Programming is part. The Vanderbijlpark Campus, with a subset of the 238 students who enrolled in 2022, is the focus of this paper. A similar, but smaller investigation was done on the data of 2021, and already reported on (Smit, 2022). The student bodies on each campus differ much in terms of computing background and access to resources, and the campus under investigation has a large percentage of students from disadvantaged communities where students typically do not have a background in computing.

The source-code detection process

Each assignment given covered a particular concept taught in the study unit addressed in a particular week. It included up to three problems for which a Python script solution had to be written. Only the scripts where plagiarism were indicated by MOSS and verified by the lecturer, were penalised.
A particular process was followed to identify plagiarism cases:

1. While marking an assignment/test, the marker would pick up SCP in a very small percentage of cases.
2. A list of potential plagiarism cases was compiled by MOSS, by indicating similarity percentages between scripts.
3. The code of each script, compared to a similar script was scrutinised to determine whether a candidate should be categorised as dishonest, and the lecturer made a final decision regarding the

inclusion of a particular case. A list of cases was announced on eFundi (the NWU's LMS). These candidates forfeited the marks for the part of the assignment plagiarised.

4. Students were allowed to state their case. It was, for instance, possible for a student to explain their approach in solving the problem identified as a plagiarism case. In cases where a student convinced the lecturer that the work done was their own work, the mark was reinstated.
5. The eventual plagiarism cases were the students who either did not complain about their status or did not convince the lecturer of their honesty.

Due to the time constraints associated with sit-down formal assessments. it was difficult for students to plagiarise code, and not many plagiarism cases were identified.

Making sense of students' learning journey

The focus of the study reported on in this paper, is on the situation created by COVID-19, specifically in 2022. To accommodate the COVID-19 restrictions, and at the same time afford students new to programming access to the lecturer and tutors, several measures were taken. A weekly theory class was conducted on the Zoom platform, then a weekly practical class focused on the demonstration of the concepts taught were also conducted on Zoom, a practical class was conducted physically in a computer laboratory, and tutor-led midnight and noon question-and-answer-sessions were scheduled on Zoom. In addition, Telegram groups were formed with the purpose to supply continuous support in addition to the interaction allowed on the learning management system (LMS) platform, which afforded access to slides, videos, a custom-made textbook, and all assessments. Due to COVID-19 restrictions, which limited the number of students allowed in a physical venue, students were encouraged to attend only the practical session if they had a pressing need to have a contact class.

The aim of the open-ended questionnaire was to form an understanding of the opinions of computing students on SCP, their programming background, their knowledge of plagiarism, and what caused them to copy code. In all subject modules at the university, the issue of plagiarism is addressed, of which students are made aware. In the context of writing code, the internalisation of its understanding is not as straight forward as when text is involved. The open-ended questionnaire (see Table 2) used, was made available to all students and not only to students who had been identified as dishonest. Participation was voluntary. Questionnaires were answered online, and results were anonymous. A total of 53 students out of 238 (22.3%) enrolments at the peak of the semester answered the questionnaire, which allowed for multiple perspectives.

Table 2:
Open-ended questionnaire used in the study

Section A: Establishing the students' programming background.	
Question 1	What is your programming background?
Section B: Finding out what the participants know about code plagiarism, whether they have been found guilty of code plagiarism and what cause students to copy code.	
Question 1	Your understanding of when one would be guilty of code plagiarism?
Question 2	Was one (or more) of your scripts identified as a plagiarism case?
Question 3	Depending on your answer above, why did you, or did you not copy code?
Question 4	Elaborate on the reason why you copied code or did not copy code?

Establishing the students' programming background reveals that 10 of the 53 participants have a background in programming (18.9%) – either having done it in high school, were self-taught, or did an online programming course. Four of the students out of the 10 (40%) who have a background in programming were identified as plagiarism cases. The remaining 43 participants (81%) have no background in programming and only started programming when starting the introductory course at university. This is a high percentage, and it correlates with the fact that most students from this environment come from schools in disadvantaged communities where they did not have the opportunity to learn to code. Twenty-nine of these students were identified as plagiarism cases (67.4%). This may explain why students who do not have a background in programming, resort to copying code either from a peer or from the other sources – they lack the confidence to code from the knowledge they are accumulating during their lectures. Numbers discussed are reflected in Table 3

Table 3:

Code explanations, response examples and occurrences

Code	Description	Occurrences	Response example(s)	Occurrences	Response example(s)
		Yes		No	
PRG_BG	Programming background	10	'I did Information Technology (IT) at school level' 'I am self-taught'	43	'None, I studied programming for the first time in CMPG111'
DIS_CS	Plagiarism case	34	'I did not understand the work and had difficulty learning the concepts' 'I didn't copy code from anyone, we were just going through the assignment as a group and then come up with an idea then we used that idea as a group'	19	'I must do my own work' 'I helped individuals who would come to me, but I now know how to move around that for the future.'
SCP_GLT	I am guilty of copying code	12	'Yes'	40	'No' 'Yes; it was not identified, but I did make use of code plagiarism'

From the question 'understanding of when one would be guilty of code plagiarism?', knowledge on plagiarism was reflected as:

- Ask another person to write code as an answer to an assessment on your behalf, using a peer's code as an answer to an assessment, and submitting it as your own answer.
- Watch a video on the Internet and using the code from the video.
- Find code on the web, or in a textbook (that is not prescribed) and using the code as an answer to an assessment.
- Collaborate with a peer (or a group of peers), coming up with a script as an answer to an assessment, and submitting it as your own answer.

From the responses above, it is evident that students know what plagiarism in this context is. Care is taken to make students aware of plagiarism in all their subject module contexts. This is also the case in the subject module under discussion, where both text and source-code plagiarism are applicable. The researchers elected to disregard a lack of knowledge on plagiarism as a cause of SCP because the students were able to select the most appropriate explanation of what it is, and therefore know what it implies. Another reason to disregard lack of knowledge on plagiarism as a cause of SCP, is because none of the students stated a lack of knowledge on plagiarism as a reason for them to copy code. This is why some of them refrain from SCP – as they do not want to be identified as plagiarism cases, while others use an opportunity to copy code even though they know it is not ethical to do so. It is possible that they think that they will not be identified as dishonest.

The question on whether students had been identified as a plagiarism case revealed that 33 of the 53 students were identified as plagiarism cases (62.3%).

The success of students

Regarding the impact that SCP had on the technical skills of students, the students found guilty of plagiarism during formative assessments - in the process discussed earlier - were tracked to determine their success in the module. Upon analysis of the data, it was found that a total of 133 students out of the 216 students participated in programming assessments, to complete the module, were identified as plagiarism cases (62%). This is a high percentage.

It is important to recognise that the SCP group of students is made up two subgroups. This distinction is supported by the data, where a fairly large number of 97 students passed the module (73%). Further analysis of this first group revealed that seven of these students obtained a mark that is 70% or higher. Twenty-five of the students obtained a mark within the range of 60% and 69%. These results indicate that almost a third of the students that were identified as dishonest were successful in passing the module with an above average mark. These students can be classified as those who allowed peers to copy their work, worked in groups, or assisted other students.

Seventy-one students passed with a mark that is between 50% and 59%. Of these, 16 passed only on their second attempt. After all examination opportunities were conducted, 36 students found to be guilty of SCP, failed the module (27.1%). This is slightly lower than the class rate of around a third. Fourteen of these students did not even qualify to write examinations (38.9%). These students can be classified as those who received help from peers – because they are not comfortable writing their own code. Although students are allowed to continue with follow-up modules in programming with a final mark of 40%, the module still needs to be passed upon a future enrolment.

Source-code plagiarism detection tools

Source-code plagiarism is a long-standing issue in the academic computing realm (Cheers, Lin & Smith, 2021). Although there are many tools available to detect SCP, limitations to each tool, such as the programming languages it is designed for, may be a disqualifying factor. To guide this process, the work of Martins et al. (2014), who developed a set of questions to determine the key characteristics of each tool is used, as shown in Table 4.

Table 4:
Code plagiarism key features (Martins et al., 2014)

Criteria	Question(s) to be answered
Open or closed source	Is the tool free to use or do you need a subscription to use the tool?
Key methods used by the tool	What methods does the tool use – to detect code plagiarism?
Different languages the tool supports	What programming languages does the tool support?
Usability of the tool	Is the tool easy to understand and use?
Online databases scanning	Does the tool make use of online databases to identify code plagiarism?
Code plagiarism report	Is the report given by the tool easy to interpret? Does it show all the necessary information, such as the part of the scripts that was copied from a peer or online source? Does it include additional features, such as … ?

In this section, the aim is to determine the capabilities of SCPD tools in the Python context. As a first cycle, the key features, as indicated above, of several tools are compared, without focusing on its support of Python. As a second cycle, the focus moves to tools supporting Python, and supplying the proof needed in an academic context. Thirdly, the suitability of each tool is used to rank the tools.

Cycle 1: Key features of open-source SCPD tools are that they are free to use, and their source-code is available to scrutinise or use. Martins et al. (2014) found that open-source SCPD tools mostly make use of three methodologies to detect copied code, namely: comparisons based on metrics that is generated from source-code (attribute-based), fingerprints that is created from source-code and then tokenised and hashed (token-based), and an internal intermediate representation created from the original code (structure-based). The open-source tools included in Table 5 are Sherlock, MOSS, and Marble. In contrast, closed-source SCPD tools are subscription-based, and the source-code is hidden to protect the intellectual property of the developer. The latter will typically allow a user to use the tool for a limited period and/or with limited capabilities – for evaluation purposes. Therefore, it is difficult to exactly explain the algorithm used in each of these tools. To explain the different methodologies, two closed SCPD tools, namely CopyLeaks and Codequiry are included in Table 5.

Cycle 2: Of the five SCPD tools listed in Table 5, only Marble does not support Python, 'although /the language-dependent part is the normalisation phase, which can easily be adapted for similar programming languages' (Hage, Rademaker & Vugt, 2010:6). Regarding the reporting focus, the fact that Sherlock and Marble do not supply proof of similarity as part of their functionality and, therefore, need a manual comparison, disqualify them from the subsequent list. This leaves us with MOSS, CopyLeaks and Codequiry.

Cycle 3: Only the subscription-based tools support online databases scanning, which is an important feature in the context of first year programs. So, CopyLeaks and Codequiry have easy-to-use graphical user interfaces (GUIs) and do well at detecting SCP, not only peer-to-peer, but peer-to-online sources as well. With MOSS there is no GUI interface, but it does well at detecting SCP, unfortunately, only peer-to-peer cases. This leaves us with the subscription based SCPD tools ranked above the freeware option.

Table 5:
Cycle 1: Code plagiarism key features on five SCPD tools

SCP detection tool (source)	**Developer (year)**	**Supported programming languages**	**Methods used to identify code copying**	**Reporting focus**
	open source			
Sherlock **(Shahabi, 2012)**	University of Warwick (1999)	Limited; Natural language, Java and C++.	Token-based. Algorithm uses incremental comparison. It compares digital signatures of files to each other.	A GUI interface makes its use seamless. Report shows that there is SCP by indicating a similarity percentage, and therefore it cannot show similarities in the code.
Marble **(Hage et al., 2010)**	Utrecht University (2002)	Java, Perl, C#, PHP, and XSLT.	Structure-based: Makes use of normalisation in two different ways and then compares the normalised files with one another.	Report only shows the percentages of code that is copied (it cannot show similarities in the code).
MOSS **(Hage et al., 2010)**	Stanford university (1994)	Supports more than 20 programming languages.	Token-based. Uses a fingerprinting technique called winnowing - which makes comparing documents much faster (Yang, 2019).	Report indicates percentages of code that is copied and highlights the code that is similar. Code is compared to code of peers uploaded for comparison.
	closed source			
CopyLeaks **(CopyLeaks, 2021)**	Yehonatan Bitton (2013)	Supports more than 20 programming languages.	Combines algorithms and artificial intelligence-based technology to detect SCP.	Utilises a GUI interface. Report is in-depth and includes figures showing percentages of code that is copied and where code is copied from (beyond the code of peers).
Codequiry **(Codequiry, 2021)**	Codequiry (2018)	Supports more than 20 programming languages.	Uses artificial intelligence machine learning on algorithms to detect SCP. The tool learns and adapts to students trying to work their way around the tool.	Utilises a GUI interface. Report is in-depth and supplies figures showing percentages of code that is copied and where code is copied from (beyond the code of peers).

FINDINGS

The four aspects focused on in this paper, are outlined below.

Source-code plagiarism detection process: With this process being time intensive per definition, a few guidelines may streamline the effort:

- Making students aware of the problems associated with SCP from the very start, will help to curb its occurrence
- Ensuring that students only submit the latest .py solution scripts will limit comparisons to only the necessary
- Structuring group work constructively
- Giving immediate feedback on plagiarism cases may convince students to rather spend the time to do their own work.

Eventually SCP needs to be managed, no matter how time intensive the process is, to ensure that students learn constructively to acquire the programming skills they would need in the corporate world. Although students are made aware of plagiarism, allowing them to experience its effects practically contributes much to help them to internalise their understanding. An added benefit is that an opportunity is created to apply the ethical concepts studied in a practical way.

Students' learning journey: When studying the feedback from students, it is clear that the issue of SCP is not black and white but nuanced. It occurs on a spectrum with outright copy-and-paste on the one side, (get) help (from) a peer or working together with peers to make sense of the material and how to tackle problems, to working alone and solving the problems oneself on the other side of the spectrum. The focus of future offerings should be to manage the first two options in such a way that students learn constructively. This will be possible in the post- COVID -19 environment of 2023, where practical classes may be conducted in a way where students work in a controlled environment where they get support while making sense of the work in a practical setting.

The success of students: For people to become good programmers they need to get as much programming practice as possible. When students are guilty of SCP, it means that they are not getting sufficient practice, which implies that they are not able to improve their programming skills. They end up failing programming modules because during formal tests and examinations they are not skilled enough to write programs according to specifications within required time limits. These students should be identified and guided to learn constructively.

Source-code detection tools: The subscription-based tools (CopyLeaks and Codequiry) were rated better than MOSS because of their easy-to-use GUIs, and support of online databases scanning. Unfortunately, with the challenges associated with COVID-19 still new, the NWU does not currently formally employ a tool to combat SCP as part of the repertoire of tools at academics' disposal. Turnitin is subscribed corporately to manage text-based plagiarism, and there is an investigation to determine the options in terms of throwing the plagiarism net wider. This leaves the academics deeming SCP a problem to be managed with MOSS at this point in time. In future the cost structure employed by CopyLeaks and CodeQuery will determine which one will be the best option for utilisation by academics.

CONCLUSION

The nature of this paper allowed the researcher to reflect on the second iteration of teaching Python, a subject module included in the syllabus as recently as 2018. Before the onset of the COVID-19 pandemic, the identification of SCP cases was not prioritised since contact classes and conducting sit-downs for major tests, and examinations did not afford students with opportunities to copy code. COVID-19 caused a large change in the educational environment, and many students saw an opportunity to complete assessments in new ways, among these copying code from peers throughout all opportunities of online (and continuous) assessment.

The outcomes of the research will direct the 2023 offering. The module offering already progressed from being an online continuous assessment module in 2021 to being online with formal sit-down assessments in 2022. The new 2023 offering is planned to be face-to-face, with formal sit-down assessments. Obviously, all the new approaches conducted during COVID-19 will be used to support this conventional approach. In the long run, the aim is to identify source-code plagiarism in the same way that text plagiarism is done, with sophisticated tools that are integrated with the learning management system which automatically warns academics of suspected plagiarism, using either CopyLeaks or Codequery. Also making students aware will assist in facilitating their internalisation of what plagiarism in this context practically means. With tools such as ChatGPT made available recently, this will become a necessity.

Creating awareness of source-code plagiarism by detecting it, is important, and this process is to be refined according to the findings to ensure that students internalise the meaning of plagiarism of code as they learn to code. This process will be facilitated by integrating the software recommended with existing university systems. Based on the knowledge gained on students' understanding of plagiarism when they start the course, amendments to the theoretical material guiding them is needed.

Therefore, future research should include interviewing students found guilty of SCP to determine their point of view and determining whether there are ways that the lecturer can support them to not revert to SCP, establishing the effect of SCP on student progress beyond the initial semester, and the effect of the implementation of improved offerings on SCP. In addition, it is the intention of the researchers to determine the magnitude of SCP across groups on the same campus, and across campuses. To support the identification of SCP, the use of Python script assessment software may be investigated to support both assessors and students. There is also potential for this research to reach beyond junior students, and look at SCP scenarios applicable to senior students, and as such, make academics and students aware of referencing code obtained from a variety of sources.

REFERENCES

Bhosale, U. (2022). Best Plagiarism Checker Tool for Researchers- Top 4 to Choose From! Retrieved 2 February 2024 from https://www.enago.com/academy/best-plagiarism-checker-tool-for-researchers/

Cheers, H., Lin, Y., & Smith, S. P. (2021). Academic Source-code Plagiarism Detection by Measuring Program Behavioural Similarity. *School of Electrical Engineering & Computing,* 1-18.

Codequiry. (2021). The effectiveness of Codequiry's code plagiarism checking tool. Retrieved 2 February 2024 from https://codequiry.com/code-plagiarism-checker

CopyLeaks. (2021). Origin of CopyLeaks. Retrieved 2 February 2024 from https://copyleaks.com/about-copyleaks

Cosma, G. & Joy, M. (2008). Towards a definition of source-code plagiarism. *IEEE Transactions on Education, 51*(2), 195-200.

Erkaya, O. R. (2009). Plagiarism by Turkish students: Causes and solutions. *Asian EFL journal, 11*(2), 86-103.

Erlingsson, C. & Brysiewicz, P. (2017). A hands-on guide to doing content analysis. *African journal of emergency medicine, 7*(3), 93-99.

Gomes, A. & Mendes, A. J. (2007). Learning to program - difficulties and solutions. *International Conference on Engineering Education – ICEE, 7,* 1-5.

Hage, J., Rademaker, P., & Vugt, N. (2010). A comparison of plagiarism detection tools. *Department of Information and Computing Sciences, Utrecht University. Utrecht, The Netherlands*, 28(1), 1-10.

Helminen, J. & Malmi, L. (2010). *Jype-a program visualization and programming exercise tool for Python.* Paper presented at the Proceedings of the 5th international symposium on Software visualization. Salt Lake City, Utah, USA. 25 October.

Joy, M., Cosma, G., Yau, J. Y.-K. & Sinclair, J. (2010). Source-code plagiarism—a student perspective. *IEEE Transactions on Education, 54*(1), 125-132.

Joy, M., Cosma, G., Yau, J. Y.-K. & Sinclair, J. (2011). Source-code Plagiarism—A Student Perspective. *IEEE Transactions on Education, 54*(1), 124-132.

Karnalim, O. (2017). Python source-code plagiarism attacks on introductory programming course assignments. *Themes in Science Technology Education, 10*(1), 17-29.

Lambert, K. A. (2018). *Fundamentals of Python: first programs*: Cengage Learning.

Maharajh, L. R. (2020). Exploring university students' perceptions of plagiarism: a focus group study. *Proceedings of the International Conference on Multidisciplinary Research,* Moka, Mauritius.

Martins, V. T., Fonte, D., Henriques, P. R. & Cruz, D. d. (2014). Plagiarism Detection: A Tool Survey and Comparison. 144-158. https://doi:

Maryono, D., Yuana, R. & Hatta, P. (2019). *The analysis of source-code plagiarism in basic programming course.* Paper presented at the Journal of Physics: Conference Series 1193 https://iopscience.iop.org/article/10.1088/1742-6596/1193/1/012027/pdf

Miller, D. T., Visser, P. S. & Staub, B. D. (2005). How surveillance begets perceptions of dishonesty: the case of the counterfactual sinner. *Journal of Personality Social Psychology, 89*(2), 117.

Myers, M. D. (1997). Qualitative Research in IS. *MIS quarterly, 21*(2), 241-242.

Newton, D. (2020). Another problem with shifting education online: A rise in cheating. *The Washington Post*. Retrieved 2 February 2024 from https://www.washingtonpost.com/local/education/another-problem-with-shifting-education-online-a-rise-in-cheating/2020/08/07/1284c9f6-d762-11ea-aff6-220dd3a14741_story.html

Newton, P. M. & Essex, K. (2023). How common is cheating in online exams and did it increase during the pandemic? A Systematic Review. *Journal of Academic Ethics*, *2023*, 1-21.

Ngo, M. N. (2016). Eliminating plagiarism in programming courses through assessment design. *International Journal of Information Education Technology, 6*(11), 873.

Shahabi, M. (2012). Comparing Three Plagiarism Tools (Ferret, Sherlock, and Turnitin). *Mitra Shahabi International Journal of Computational Linguistics (IJCL), 3*(1), 53-66.

Smit, I. (2022). The Value of the Measure of Software Similarity Tool in the Assessment of Introductory Programming-Making Sense of a Changing World. *Proceedings of the International Conference on Teaching, Assessment and Learning in the Digital Age*, Durban, South Africa.

Thurmond, V. A. (2001). The point of triangulation. *Journal of nursing scholarship, 33*(3), 253-258.

Vom Brocke, J., Hevner, A., Maedche, A. (2020). Introduction to Design Science Research. In: Vom Brocke, J., Hevner, A., Maedche, A. (Eds) *Design Science Research. Cases. Progress in IS*. Springer, Cham. https://doi.org/10.1007/978-3-030-46781-4_1

Yang, D. (2019). How MOSS works. Retrieved 2 February 2024 from https://yangdanny97.github.io/blog/2019/05/03/MOSS

Younas, R. (2021). List Of Best Code Plagiarism Checkers In 2021. Retrieved 2 February 2024 from https://ssiddique.info/list-of-best-code-plagiarism-checkers.html

Zarfsaz, E. & Ahmadi, R. (2017). Investigating some main causes and reasons of writing plagiarism in an EFL context. *International Journal of Applied Linguistics and English Literature, 6*(5), 214-223.