Eliminating Plagiarism in Programming Courses through Assessment Design

Minh Ngoc Ngo

***Abstract*—In recent years, plagiarism has become increasingly prevalent in programming courses. As a consequence, there is an enormous amount of research work in the area of preventing and detecting plagiarism. There is a number of tools that are available for automated detection of plagiarism. However, most of the cases must be reviewed by instructors to confirm the conclusions suggested by the tools. This is a labour intensive process which always requires too much time and effort from the instructors. More importantly, it must be highlighted that it is always possible for undetectable plagiarism to occur, no matter how sophisticated the tools available. In this paper, we propose an assessment design method that aims to prevent students from plagiarizing without deep understanding and to promote peer learning. We present our sample design and share some initial results in implementing the method in a programming course.**

***Index Terms*—Plagiarism, programming, assessment design, pedagogy.**

I. Introduction

Joy *et al.* [1] defined software plagiarism as the act of "unacknowledged copying of documents or programs". With the increasing popularity of programming courses, either within a computing related degree or as part of another degree, software plagiarism has becoming a big concern for academic institutions. Assessment of programming courses usually involves students writing programs for some assignments, which are then marked against criteria such as coding style and program correctness. Unfortunately, due to the electronic nature of these programming assignments, it is very easy for students to exchange copies of source code they have written or obtained from other sources.

An enormous amount of research work has been published in the area of preventing and detecting plagiarism in programming courses. However, due to the large number of students in a programming course, it is often difficult to detect plagiarism. Moreover, when student's knowledge become more advanced, plagiarism becomes more subtle and elaborate. Therefore, it is even more difficult to detect for the class instructors [2]. Even though there are some tools available to enable automated detection of plagiarism, most of the cases are reviewed by instructors to confirm the conclusion suggested by the tools. This is a labor intensive process which requires much time and effort from the instructors. It must be highlighted that we could never detect 100% of all the plagiarism cases, no matter how sophisticated the tools available [1]. Another problem which makes detection of source code difficult is that similar coding can be used for the same application [3]. This is mainly due to the assignment design. When instructors design an assignment, which is very close-ended, chances are it is very hard to tell whether students plagiarize or they just happen to have the same solutions. For a simple example, it is very likely that instructors will get similar code from students if they are asked to write a program to swap two integers. Therefore, the plagiarism detection technique must take into account "which of the different conditions that define a certain specific situation allow the case to be considered as plagiarism" [2].

Plagiarism is often considered serious by all academic institutes mainly because it prevents students from achieving the main goal of programming: to interpret source code and to write source code [4]. According to Clough [5], when students plagiarize, they do not benefit from the experience of writing code, which is considered the most effective way to learn programming. Even more seriously, students might alter the plagiarized code to make them appear unique without fully understanding the code.

A. Why Students Plagiarize?

Various reasons why students plagiarize have been highlighted by researchers [1], [4] include:

1) **Solution availability**: Students tend to plagiarize if solutions to assignment can be easily obtained from Internet or similar sources.
2) **Procrastination**: Steel [6] reported that 80% to 95% of college students procrastinate, particularly when it comes to doing their course work. Procrastination is usually due to underestimating the amount of time that one needs to spend on coursework. It has become a bad habit which leads to doing rush works at the last minute. Harris [7] suggested that instructor should set "intermediate" due dates where sub tasks of a large assignments are due. In this ways, students are less tempted to plagiarize to meet deadline.
3) **Motivation**: Poorly motivated and week students copies then edits a friend's program or solution obtained from other sources, with or without permission, to minimize the work required. In this case, students usually hope that the unacknowledged copied work with go unnoticed. Some students have the perception that it is more beneficial to "breeze through assignment than to learn from them" [4]. Some other students do not feel confident of their own ability to complete assignments. Harris [7] also stressed that instructor should help students to realize that assignments are designed to help them to learn and gain knowledge. It does not make sense to plagiarize to get good marks and pass the course without

Manuscript received May 10, 2015; revised July 21, 2015.

Minh Ngoc Ngo is with Singapore Institute of Technology, Singapore (e-mail: cristal.ngo@singaporetech.edu.sg).

actual knowledge and understanding. Yet, based on our experience, no matter how much instructors emphasize this to students, if assignment is not well designed, plagiarism is not avoidable.

B. Work-Around Solutions to Plagiarism

To the best of our knowledge, there is no absolute technique or tool to detect plagiarism, both manually and automatically. Abraham [4] summarized methods to help in preventing and partially detecting plagiarism which include 1) plagiarism education, 2) modifying course grading structure, and 3) using plagiarism detection software.

Students should be clearly informed of what constitutes plagiarism. In our experience, we give briefing to students at the beginning of each semester during orientation day. We show students concrete examples of plagiarized code. Each student must sign attendance as a form of confirming that they are aware and understand about this problem. Instructors should also show students how plagiarism can be detected using existing tools. The effectiveness of this approach is, however, not proven.

Bowyer *et al*. [8] suggested that one of the main reason that students plagiarize is because they want to achieve higher grade without spending much effort and time on it. As such, non-assessed assignments could be given to students and the course grade depended only on in-class activities such as exams. However, it has also been proved that many students are not motivated to do this kind of assignments; although the assignments "help them to prepare for quizzes and other in-class activities" [8]. Our proposed strategies follow this method of "no incidents of plagiarism". However, our assignments are designed carefully to avoid copying without understanding and our in-class assessments are designed to test students' understanding on their own source code. As such, students could not pass the in-class assessments without attempting the assignments.

Many complex tools that have been developed to automate much of the plagiarism detection process [9]-[12]. Most of the tools can detect plagiarism, however, some tools are designed to target certain patterns of plagiarism and are not suitable for other kinds. MOSS [12] ignores "comments and identifier names" thus focusing more on the structure and pattern of the program. However, obvious cases of plagiarism detected through identifier and comments cannot be detected efficiently with MOSS. Some applications, such as MOSS [12], compare submitted code to other submitted code without building a database of code. Therefore, it is impossible to detect if students copy source code from students in previous batch. Some tools take significant amount of time to analyze source code. Code Match [11] uses a combination of algorithm; thus it is reliable but could take hours or days to complete the detection process.

In this paper, we propose a method to avoid plagiarism without understanding through assignment and assessment design. We believe that learning from worked examples, including from friends or other sources, is useful. The key point is, however, students must understand the examples clearly and be able to modify the examples to meet new requirements. In our programming assignment design, source code skeletons are designed to force students to follow some certain program structure. As such, it is impossible for students to copy source code from other sources because students need to adapt the existing source code into the code skeletons' structure.

We also design assessments to make sure that students do not copy source code from their friend without understanding the source code. Students are not graded based on their source code but their understanding of the source code. More specifically, how well they could modify their source code to meet new programming requirements given in the assessments. In this way, students are encouraged to learn from their peer. More importantly, students are not afraid to teach their peer and share with their peer their understanding and knowledge.

In the remaining sections, we introduce our assessment design strategies and samples to address the above challenges. Finally, we share our positive experiences in implementing this method.

II. Assessment Design Strategies

Our objectives in coursework design include 1) covering the learning outcomes, 2) helping students to learn programming and 3) promoting peer learning among students. The first objective is easy to achieve. The second objective can be achieved given that one must ensure that students did not plagiarize without fully understanding. The third objective is not easy to achieve. If instructors using plagiarism detection techniques seriously, either manual or automated, it is very likely that students will not share their ideas with others because they are afraid that their idea will be plagiarized. In our opinion, students learn best from their peer. This drives us to design an assignment and assessment framework to promote peer learning and yet still ensure students' understanding of the subject matters.

We are also inspired by learning from "worked examples", which has recently attracted much attention [13]. Researchers have found that worked examples provide good opportunities for learners to study and emulate. Brusilovesky [14] proposed the WebEx platform to enable learning from examples in programing courses. According to Brusilovesky, learning from worked examples is especially useful in the domain of programming. "Both experienced and novice programmers often use program examples they have created or solved from the past to solve new programming tasks" [14]. In our opinion, it is also important that students master a set of system skills to be successful in professional development. Beside development, students must also be able to comprehend and maintain existing applications, open for adaptation to new needs and for reuse some of their components. As such, it is good if students are able to understand existing applications and adapt them for their own purpose. However, the key here is we must ensure that student understand the "worked examples".

Our assessment design strategies consist of two key components 1) assignment design with *code skeleton* and 2) in-class assessments based on assignment content. Fig. 1 shows the whole assessment flow. We will elaborate more on this in the subsequent sub-sections.

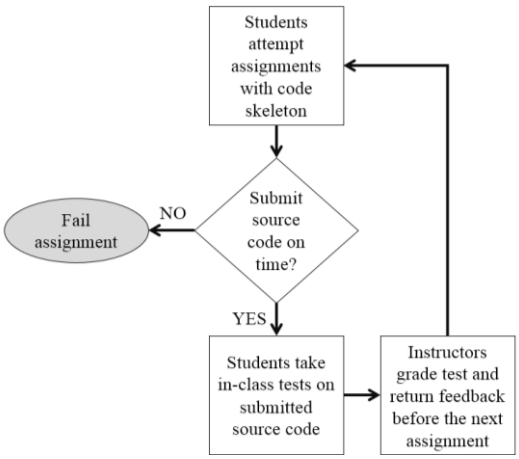

Fig. 1. Assessment flow.

A. Assignment Design to Avoid Plagiarism without Understanding

The key in our programming assignment design is a *code skeleton* provided to students together with the requirements. For example, in one of the programming assignment, we asked our students to implement a Sudoku game in C programming language.

Numerous solutions can be found for this problem on the Internet. Therefore, we designed the code skeleton to force students to write programs which must follow some certain structures, it could simply be data structure or it could be some predefined functions that they must use. Fig. 2 gives a sample of the code skeleton that was given to students to complete this task. The constraints include students must 1) use all the functions in the code skeleton, 2) not remove any code in the code skeleton and 3) construct the Sudoku board using the board 1-dimensional array. The design of this code skeleton is based on the observation that most of the implementations of this game use 2-dimensional arrays. As such, even if students obtain solutions from other sources, they must study the source code and rewrite most of them to fit into the given code skeleton.

There are several benefits to provide students with a code skeleton:

1) Students develop the ability to understand existing source code and how they should write their source code to fit into an existing structure.
2) The code skeleton gives students who are new to programming some ideas to start.
3) The code skeleton prevents students from directly copying code from the Internet. Even if students find solutions from the Internet, they cannot copy and paste the source code directly into the code skeleton. This is because the code skeleton has been designed such that there are certain data structures, variable names that students have to use in their implementation. As such, in most cases, if they copy the code directly, the program will not compile. Instead, to re-use solutions from the Internet, students need to understand the solution and translate them to fit into the code skeleton.

B. Assessment Design to Promote Deep Understanding and Peer Learning

With code skeleton, we can ensure that students do not directly from other sources without fully comprehending the source code. However, we still need to design assessments to ensure that students do not plagiarize from their peer and to encourage peer learning. The main features of our assessment design to address these challenges include:

1) Weekly/fortnightly assignment submission. Students are required to submit their source code at the end of each week. Having weekly or fortnightly assignment minimizes students' procrastination because it is easier for them to manage their time for a week. They are more motivated to complete their lab project when knowing that the deadline is at the end of the week.
2) An in-class test after every one or two programming assignments. The test motivates and creates pressure for students to complete the assignment on time. Constructive feedback and scores are returned to students before the next test so they can learn from their mistakes. Source code must be submitted before the class test. Table I gives a sample of the timing of assignment submissions and in-class test. Even though students are not graded based on the submitted source code we will not grade their lab tests if the source code are not submitted on time. Fig. 1 illustrates in detail the assessment flow.

TABLE I: ASSIGNMENT AND TEST TIMING

Assignment	Deadline	Test
Assignment 1	9AM, 27 April 2015	10 AM, 4 May 2015
Assignment 2	9AM, 4 May 2015	

```
#include "Sudoku.h"
#include <windows.h>
#include <stdio.h>
#include <conio.h>

int board[BOARD_SIZE*BOARD_SIZE] = { EMPTY };
HANDLE h;

void menu()
{
    int choice;
    initialize_screen();
    system("cls");
    printf("\n--------SUDOKU MENU--------");
    printf("\n1 : Start Game");
    printf("\n2 : Exit");
    printf("\nEnter your choice:>");
    scanf("%d", &choice);

    switch (choice)
    {
    case 1:
        initialize_game_board();
        printBoard(); /*You can remove this line*/
        startGame();
        break;
    case 2:
        exit(1);
    default:
        menu();
    }
}
```
Fig. 2. A sample Sudoku code skeleton.

After every one or two programming assignments, students are required to take an in-class test. Each test is designed to really assess students about their comprehension of their submitted source code.

In each test, we often change the original requirements for the respective assignment(s). We then ask the students to

identify the changes required in source code to meet the changes in the requirements. Students are required to bring a hard copy of their source code and make changes directly on the printed source code.

For example, Fig. 3 and Fig. 4 show one task in a programming assignment and part of the accompanied code skeleton:

In the "PLAYER_MENU", if user enters "New", the program will ask the player to enter his name. The program will then automatically generate a unique ID and add his record to the array playerList.

The uniqueID should consist of the player name's first 3 letter, 2 generated digits and the last letter in the player's name. You should think about how to generate the 2 digits. Function int checkUniqueness(char * idString) can be used to check whether idString is unique. It returns 0 if idString is not unique, returns 1 otherwise. To do this, you should insert your own source code into the following two places:

```
/*insert your own code here (2)
  This code segment should call to
  checkUniqueness() to check for the
  uniqueness of the generated ID */

/*insert your own code here (3)*/
```

Fig. 3. Sample assignment task.

Fig. 5 shows a sample test question that has been designed to assess the assignment task in Fig. 3. In the assignment, we asked student to "create a player ID from the first 3 characters of their full name and 2 random numbers". However, in the test, we changed the requirement to the one shown in Fig. 5. Only by truly understanding the original source code, students are able to modify the code to meet this new requirement.

The basic steps applied in designing the in-class test include:

1) Identify the learning outcomes or the objectives of each assignment.
2) Identify important code segments or assignment requirements that correspond to the learning outcomes.
3) Modify the assignment requirements identified to test student's understanding.

```
void addNewPlayer(){
    char name[20];
    void generateId(char*, char*);
    int checkUniqueness(char*);
    char id[7];

    if (numOfPlayers == MAX_PLAYERS){
        /*insert your own code (1)*/
    }
    else{
        printf("Enter your name: ");
        scanf("%s", name);
        generateId(name, id);

        /*insert your own code here (2)
        This code segment should call to
        checkUniqueness() to check
        for the uniqueness of the generated ID
        */
        numOfPlayers++;
    }
}
```

Fig. 4. Another sample code skeleton.

Assume that the requirement for the player id has been changed. The id should contain:

- 2 last characters from the player's name,
- 3 random numbers and
- the first character from the player's name.

Modify your implementation of the function void generateId(char *name, char *id) accordingly

Fig. 5. Sample test question.

Take the test question in Fig. 5 for example. In this assignment requirement, the objective is to test students on string formation and manipulation in C programming language. To be able to solve the test question, students need to perform several tasks that require deep understanding and skills to change source code:

1) Identify the code segments in their source code that implement this requirements.
2) Instead of getting the first 3 characters in a string, students need to know how to access the last 2 characters. The original requirement of using the first 3 characters is slightly easier. As such, students first need to understand the original implementation and apply the technique to get the last 2 characters.
3) Similarly, if students understand how to generate 2 random numbers, generating 3 random numbers should be easy for them.

As having been highlighted in the introduction, in traditional programming assessment design, students have to submit their source code and instructors grade the source code. In this design, instructors need to check for plagiarism. Even with the help of an automated tool such as JPlag [9], instructors still need to manually verify the results produced by the tools. Moreover, students are afraid to share their ideas and knowledge with their peer because they are afraid that their source code will be plagiarized. As such, this design does not promote peer learning. It makes it more difficult for students who have no knowledge in programming. This group of students requires a lot of support from their peer.

In our assessment design, we do not check if students copy source code. We are testing students' understanding. Students are free to use source code from the Internet or from their peer, study the source code and learn from worked examples. This is because we do not grade students based on the source code submitted. We only evaluate the students based on the test results, which students need to understand the source code really well to pass the test. Therefore, even if students are getting source code from their peer, they need to put an effort to understand the source code. As students are not assessed by the source code submitted but by the ability to understand and change the source code, students are more willing to help their peer to learn. By designing our in-class test based on the content of the programming assignments, we also avoid the problem highlighted by Barry [15], where students do not put effort into assignments that are not graded.

C. Assignment Design to Maintain the Interest of More Advanced Students

Jenkins *et al.* observed that students in a programming course are usually from different majors and background and that this diversity tended to increase over time [16]. The diverse student population creates major challenges in

teaching introductory programming courses. It is difficult to decide on the depth of knowledge to be covered in the course to suite all the students. Moreover, it is difficult to motivate the interest of students whose minds are already set on a different major [17]. As a consequence, it is very hard to maintain the interest of this group of students in a programming course.

Programming assignments can be designed with bonus tasks. These tasks are more challenging which are designed based on the basic tasks. Advanced students therefore need to finish the basic tasks before moving on to the bonus ones. For these bonus tasks, there are equivalent bonus questions in the in-class tests.

Interestingly, we observed that there are about 70% of students attempted the bonus tasks, including students who has no programming experience. Advanced students are more motivated to complete the bonus task ahead of time. They also often provide help to their peer in attempting the bonus tasks. We feel that student view the bonus tasks as some incentive or some form of back-up plan in case that they fail to answer the test questions for the basic tasks, they can attempt the bonus questions.

III. Initial Results

We have applied the assessment design strategies presented in the previous section to a Programming Fundamentals (PF) course in September 2014. The failure rate for this course as reported in January 2015 is 6.3%. Other institutions have reported failure rates for introductory programming courses ranging from 25% to 50% [18], [19]. The failure rate in our first cohort was radically lower. Surprisingly, the percentage of students who received grade A- and above is 65.7%. We strongly believe that this result is highly influenced by the assessment design which encourages peer learning.

Within 13 weeks we managed to cover all the predefined learning outcomes. Most of the learning outcomes are incorporated in our lab assignments. The last week was dedicated to revision. Therefore, practically, there are only 12 learning weeks. We designed 6 lab assignments and conducted 3 in-class tests, one test after every two lab assignments. Fig. 6 shows the students' performance after the first test. The first test was conducted at the fifth week of the course.

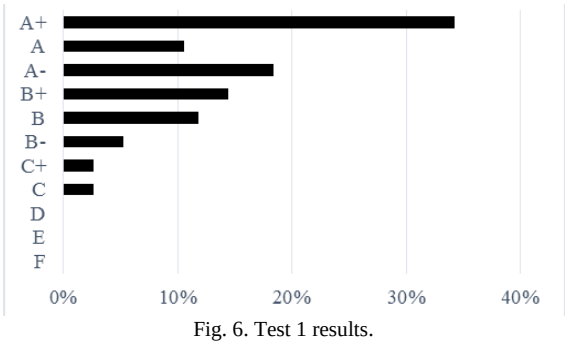

Fig. 6. Test 1 results.

According to the figure, the grade average is A- (77.2%) and the minimum grade is C. This result provided evidence to show a really good achievement in term of students' understanding on the fundamental concepts including: algorithms and problem solving, testing and debugging, functional decomposition, data types and control structures.

Starting from week 5, we taught students more complex concepts such as functions, lists, arrays and pointers. The second was conducted in week 9 to test all of these concepts. According to a survey done by Iain *et al*. [20], pointers and parameters passing to functions are some of the most difficult programming concepts to comprehend for beginners.

The graph in Fig. 7 shows the results for the second test. Surprisingly, the class grade average increases to A- (75.67%). There are 2 students that failed the test (grade F). However, it is encouraging that the number of students who are mastering the concepts are increasing. We believe that this is the good achievement of peer learning where students really help each other to deeply gain the knowledge.

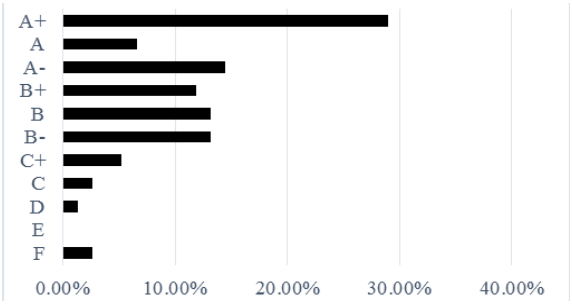

Fig. 7. Test 2 results.

Fig. 8 shows the last test results. The last test was conducted in week 13. In the last four weeks of this course, we taught students even more complex concepts in programming which require a deep understanding of pointers including strings, files, and advanced data structures such as linked list. We did not expect that students' performance would be as good as in the second test because they had many assignments in other modules due in the last week of the semester. However, as shown in Fig. 8, the number of "A+ students" increases as compared to the second test. The class average increases to A- (77.97%).

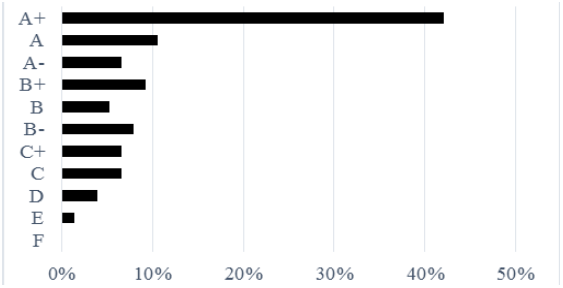

Fig. 8. Test 3 results.

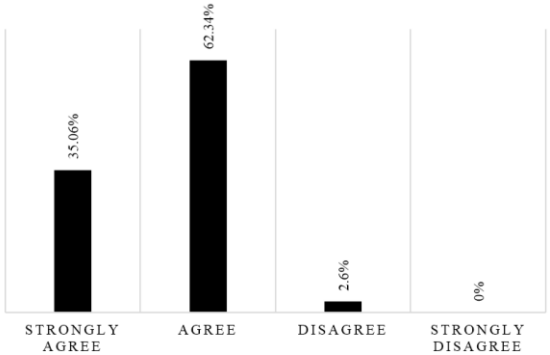

Fig. 9. "The knowledge gained is practical and useful."

Students were surveyed during the last week of the course. The survey results indicate that 100% of students agree that "the module was covered adequately according to the syllabus". According to the graphs in Fig. 9 and Fig. 10, 98 % of our students agreed/strongly agreed that "the knowledge gained is practical and useful" and about 82% of our students said that they have increased their competency as a result of taking this course.

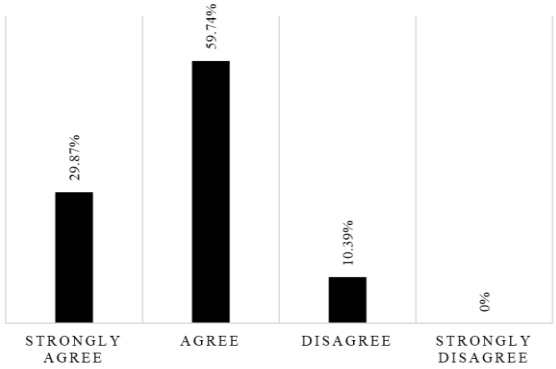

Fig. 10. "I have increased my competency as a result of taking this course/subject."

Survey responses also suggest that students enjoy the experience they had in this course. For example, students were asked to "give responsible and constructive feedback regarding what you liked about the course". Students responded with enthusiastic feedback as shown in Fig. 11.

We feel that with this approach to assessment design, students are more responsible for their own learning and understanding. They really need to understand what they are doing and why they use a certain method to solve a problem. On the other hand, instructors are relived from manual plagiarism detection and review. With careful assignment and test design that are aligned with the learning outcomes, instructors can ensure that students really understand and be able to apply taught concepts.

However, we believe that the best achievement from this assessment design strategy is the active peer learning that can be observed from the students. As we have mentioned in the previous section that due to the assessment design, students are willing to help their peer to learn. Students automatically form study group, actively discuss about the lab problems that they need to solve and independently write their source code to make sure that they understand their own code to pass the lab tests. Students also share their ideas freely on the school's forum and help each other to correct their understanding.

"Introduction to C is interesting. The method of teaching is a lot different."

"I probably learnt more (and understand more) about pointer here than I ever did in my poly."

"We are provided with sufficient opportunities to apply learning in lab exercises."

"I like the independent learning style during labs."

"The lab exercises are challenging and fulfilling."

"I understand C programming language from the very basic of that language."

Fig. 11. Students' comments on programming fundamentals.

IV. Conclusions

We have proposed an approach to design assignments and assessments in programming courses to eliminate plagiarism. In the domain of programming, learning from worked examples is especially useful where students learn to interpret existing source code and modify it to their needs. The key in learning from examples is that we need to make sure that students understand the examples, not just copy the source code for the sake of completing assignments. As such, in our design strategy, code skeletons are developed for each assignment to prevent students from copying without understanding. In-class assessments are then designed based on the assignment content to test students' understanding and ability to modify their source code to meet new program requirements.

The initial results and feedback from students show potential benefit of our design method in improving students' understanding and performance. More importantly, it eliminates instructors' time and effort in detecting plagiarism.

However, it is not easy to design a series of assignments with code skeletons and accompanied assessments. We spend a lot of time to designing the application and the code skeletons for lab assignments. The difficulty lies in designing code skeletons to test students' skills and understanding. Moreover, the assignments should not be used from one cohort to another to totally eliminate plagiarism. We are looking into assignment design patterns to enable instructors to design similar assignments but in different application domains to overcome this challenge.

References

[1] M. Joy and M. Luck, "Plagiarism in programming assignments. Education," *IEEE Transactions*, vol. 42, no. 2, pp. 129-133, 1999.
[2] A. M. Bejarano, L. E. García, and E. E. Zurek, "Detection of source code similitude in academic environments," *Computer Applications in Engineering Education*, vol. 23, no. 1, pp. 13-2, 2015.
[3] M. N. More, M. C. Patel, and M. A. Bhootra, "Plagiarism detection in source code," *International Journal for Innovative Research in Science and Technology*, vol. 1, no. 10, pp. 109 -112, 2015.
[4] S. Abraham and G. Milligan, "Software plagiarism in undergraduate programming classes," *Information Systems Education Conference*, 2008.
[5] P. Clough, "Plagiarism in natural and programming languages: An overview of current tools and technologies," *Research Memoranda: CS-00-05, Department of Computer Science*, University of Sheffield, UK, pp. 1-31, 2000.
[6] P. Steel, "The nature of procrastination: a meta-analytic and theoretical review of quintessential self-regulatory failure," *Psychological Bulletin*, vol. 133, no. 1, p. 65, 2007.
[7] R. Harris, "Anti-plagiarism strategies for research papers," *Virtual Salt*, vol. 7, 2002.
[8] K. W. Bowyer and L. O. Hall, "Reducing effects of plagiarism in programming classes," *Journal of Information Systems Education*, vol. 12, no. 3, pp. 141-148, 2001.
[9] L. Prechelt, G. Malpohl, and M. Philippsen, "Finding plagiarisms among a set of programs with JPlag," *J. UCS*, vol. 8, no. 11. pp. 1016, 2002.
[10] M. J. Wise, "YAP3: Improved detection of similarities in computer program and other texts," *ACM SIGCSE Bulletin*, 1996.
[11] B. Zeidman. (2015). Tools and algorithms for finding plagiarism IN source code. [Online]. Available: http://www.ddj.com/architect/18440573
[12] A. Aiken, *Moss (Measure of Software Similarity) Plagiarism Detection System*, 2000.
[13] S. S. Abdul-Rahman and B. D. Boulay, "Learning programming via worked-examples: Relation of learning styles to cognitive load," *Computers in Human Behavior*, vol. 30, pp. 286-298, 2014.

[14] P. Brusilovsky, "WebEx: Learning from examples in a programming course," *WebNet*, 2001.
[15] E. S. Barry, "Can paraphrasing practice help students define plagiarism?" *College Student Journal*, vol. 40, no. 2, p. 377, 2006.
[16] T. Jenkins and J. Davy, "Dealing with diversity in introductory programming," presented at 1st Annual LTSN-ICS Conference, Citeseer, 2000.
[17] M. Pregitzer and S. N. Clements, "Bored with the core: Stimulating student interest in online general education," *Educational Media International*, vol. 50, no. 3, pp. 162-176, 2013.
[18] N. Herrmann, "Redesigning introductory computer programming using multi-level online modules for a mixed audience," *ACM SIGCSE Bulletin*, ACM, 2003.
[19] N. Nagappan, "Improving the CS1 experience with pair programming," *ACM SIGCSE Bulletin*, ACM, 2003.
[20] I. M. A. G. ROWE, "Difficulties in learning and teaching programming — Views of students and tutors," *Education and Information Technlogies*, vol. 7, no. 1, pp. 55-66, 2002.

Minh Ngoc Ngo received a B.Eng in computer engineering and a Ph.D in software engineering from Nanyang Technological University (NTU) Singapore in 2004 and 2009 respectively.

In 2009, she joined the Division of Information Engineering, School of Electrical and Electronic Engineering (EEE) in NTU as a teaching fellow. From 2011 to 2014, she was a lecturer at Singapore Institute of Management (SIM) where she receives SIM lecturer of the year award in 2011 for lecturing and tutoring in the collaborative bachelor program with University of Wollongong, Australia. Since 2014, she has been with Singapore Institute of Technology (SIT) as an assistant professor. Her current research interests include software design and automated software testing, computer science education, innovative teaching and learning pedagogy and effective curriculum design.

She was a recipient of the NTU research scholarship from 2004-2008 and a recipient of the Singapore Ministry of Foreign Affairs (MFA) undergraduate scholarship for the best ASEAN students from 2000-2004.