HOW STUDENTS' PROGRAMMING PROCESS DIFFERS FROM EXPERTS – A CASE STUDY WITH A ROBOT PROGRAMMING EXERCISE

Erno Lokkila[1], Teemu Rajala[1], Ashok Veersamy[1], Petra Enges-Pyykönen[1], Mikko-Jussi Laakso[1], Tapio Salakoski[1]

[1]*University of Turku (FINLAND)*

Abstract

Programming is a creative skill. Hence, there are always multiple ways to solve a problem. Still, some solutions are better, more efficient or more elegant than others. The quality of the code can be evaluated based on multiple factors, and it strongly depends on the case which of those is the most applicable. In our university, we use an in-house developed learning environment called ViLLE. It contains several automatically assessed exercise types designed for all learning situations, with extensive support for programming exercises. In this paper, we discuss the development and application of a particular type called Robot exercise. In the exercise, the students need to program a crane to move a number of boxes into their goal positions. Since the same task can be solved with a number of programs, we analyze the students' solutions to two different cases meticulously, with additional focus on the development of the answer through several iterations. The exercise is designed to give the maximum score for the more optimized solutions. Hence, the answers are categorized based on the optimization level and the clever utilization of techniques such as repetition and methods. Based on the analysis, we finally discuss students' capabilities for problem solving and logical thinking and possible methods for increasing the level of these during the course.

Keywords: Programming education, Algorithmic thinking.

1 INTRODUCTION

Teaching programming is widely known to not be a trivial task. This is supported by a myriad of studies [3, 4, 5, 11]. The focus seems to be in studying the effectiveness of the novice programmer after and during a programming course. However, there is not as much focus on what should the students be taught.

Currently, programming courses offered at our department show students complete programs, then dissect and analyze the features and syntax of that code. Instead of merely discussing a complete program code, student would benefit more from being taught – or even shown – the process of how the result was achieved. This approach is based on the assumption that better results are achieved by more skilled or better programmers. Thus, showing what the experts have produced, students would achieve similarly. However, programming is not (all) about reading complete program code.

Programming is a process. This is often overlooked in education, where only the end result – complete program code – is analyzed. By showing students only the code resulting from the process of programming by experts, an expectation is instilled to students. This expectation is that experts somehow sit in front of the computer, then start typing and a few moments later have produced perfect, working code. It is not impossible to imagine this is in part why students often find programming discouraging; the expectation is to get the end result without the process.

2 RELATED WORKS

Computer programming is the art of developing programs to perform a specific task on the computer. It seems that writing an efficient program requires problem solving or analytic skills. Moreover, the written program must be readable and work correctly under all possible conditions [1]. Computer programming courses requirements should include a good understanding of programming concepts and metacognitive skills in order to be proficient in computer programming [12]. There have been research studies done on student programming ability, thinking skills and the way they attempt to write computer programs to solve a given problem

Problem solving is the process of finding solutions for given problems or tasks or issues. It is an essential skill and most problem solving skills are developed through everyday life and experience. Students learn problem solving skills related to the subject from the teacher, textbooks and other learning sources and convert that knowledge into usable skills to solve problems [7]. In computer programming, student need to know how to apply programming specific general skills such as generating solution to a problem, abstracting sub-problems and testing the solution to a computer problem, debugging the solution if the test disclose any errors [8].

Moreover, developing a program solution for a given task involves many activities and the solution can be written in many ways. Even with many possible solutions, some students have difficulties in converting a given problem in to a computational solution. However, measuring the quality of computer program and a programmer's ability in computer programming is a difficult task [2, 15]. Shneiderman reported that Gilb listed nearly 40 software metrics that can be used to measure the program quality, however many of those metrics were not defined with validated studies [2, 14]. In education, computer programming is one of the core skill that IT and computer science students are expected to master. Educators often would like to see improvements in the code of those students, who completed introductory programming courses as a prerequisite [3]. Still, examining the quality of student code and measuring students' programming ability is a hard task for educators. There has been research done on analyzing student's programming skill in learning FORTRAN, Java and C++ [2, 3, 11, 12]. It is identified that student who had prior experience in programming would code better than who do not [4, 13]. However, Breuker et al. research on measuring the quality of code written in Java by first year and second year computer science students reported that there is no much difference between the static qualities of code produced by both first year and second year students [3]. In addition, McCracken et al. assessed the programming skills of first year computer science students of universities from various countries, and identified that many students who completed at least two programming courses did not know how to abstract the problem from its description [9]. Lister et al. conducted a multi-national study to examine the students' reading and tracing skills in learning Java and C++ programming. They analyzed university students' multiple choice test answers, and concluded that, many students lack the ability to decompose the problem in to sub problems and implement them in order [11]. Lahtinen conducted a cluster analysis study against Bloom's Taxonomy on students' answers of C++ arrays exercises and identified six groups of students with similar skill profile that have problems in computer programming [6].

Moreover, although many studies examined the students programming ability, distinguish between junior undergraduate, undergraduate and senior undergraduate students none have yet fully compared the coding quality of university students with professional computer programmers. So, it would therefore be helpful to know how student understood the given problem and arrived at the programming solution. In addition, tracking the process of student problem solving behavior in comparison to the problem solving behavior of professional programmers would be highly helpful for educators to measure the programming ability of students.

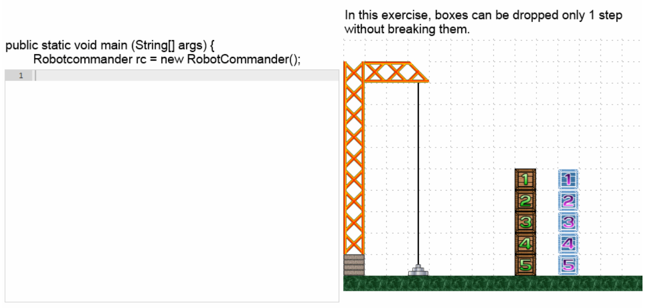
3 DATA AND METHODOLOGY

The data was collected from an introductory course to programming at the University of Turku during the fall semester of 2014. The course is obligatory to Computer Science majors and minors, and the recommended time to take the course is during the freshman year. As such, the number of partaking students was fairly high: 227 registered students. However, due to overlapping schedules for some minors, not all students were university freshmen.

Data collection was performed on the ViLLE-learning platform using the Robot Exercise available on the platform. There were two obligatory Robot Exercises on the course, however, only the data from the first one is analyzed in this paper. Both obligatory robot exercises had to be passed with at least a 50% mark to be eligible for the final exam. 30 students did not give any submissions to either Robot Exercises. The data consists of all student submissions for the assignment that were successfully compiled and run.

In the Robot Exercise, the student is presented with a stack of boxes and a crane (Fig 1.). The task for the student is to move the numbered boxes from the stack to positions marked on the screen. Students are presented a simple API to allow them to operate the crane by moving it in any of the four primary directions and grabbing and dropping boxes. Students may, at any moment, run the code and see, in the case of working code, how the code they have written works or, for uncompilable code, a standard Java compiler error.

Several different versions of the exercise were distributed to students. The exercise was varied in the distance between the stacks of boxes and the starting location of the crane so that direct copy-paste was not possible unless students first found another student with the exact same version of the exercise. As working in pairs was allowed, this type of pair-work could explain the majority of the cases where a full score was received with only few submissions.



```
public static void main (String[] args) {
      Robotcommander rc = new RobotCommander();
```

In this exercise, boxes can be dropped only 1 step without breaking them.

Figure 1. A Robot exercise

The ViLLE-platform scores submissions by first counting the correctly placed boxes and then subtracting any penalty points gathered by surplus movements of the crane and lengthy code. Thus, a submission which correctly moved three boxes out of four with somewhat lengthy code would receive a score of 65%; 75% for the boxes minus a 10% penalty. The threshold for the penalty points are set by the teacher in the exercise editor, which allows for fine-tuning the difficulty level for getting the maximum points.

Data used in the analysis consists of the submissions of the students and three professional programmers, who were given the same exercise as the students. The version in figure 1 was chosen to be done by each expert programmer. Statistical information is shown in table 1. Only submissions that consisted of code that compiled and finished (i.e. no infinite loops) were collected. This was because the focus is on reaching the solution instead of the solution itself.

Table 1 Statistical features

	Students	Experts
n	197	3
Submissions	5946	14
Average submissions	30.18	4.67
Mode of submissions	12	N/A
Median of submissions	18	6
Standard deviation of submissions	29.56	2.62
Average lines of code per submission	116.55	29.64

The total number of submissions made by the students to the exercise was 5 946. The average number of submissions made per student was 30.18. The least amount of submissions was 1, whereas the most was 209. For the expert programmers, the total number of submissions was 14, which averages to 4.67 submissions made by an expert programmer. The least number of submissions was 1 and the most 7. The average number of lines per submissions is also clearly shorter: 116.55 for students and 29.64 for the experts. Clearly the expert programmers are doing something different when solving the task when compared to the students. Table 2 shows which code features were found in the submissions.

Table 2 Percentages of programmers using code features

	Students	Experts
Branching (if, switch)	42.25	100
Methods	85.92	100
Arrays	9.15	33
Comments	38.03	66
Loops	93.66	100

Even if the number of experts is rather low compared to the number of students, a trend seems clear: experts utilize the basic control structures (branching, loops) and methods even when solving simple tasks. This is not so for the more inexperienced students. Less than seven percent of students clearly attempted a brute force solution, which never contained any loops whatsoever. Less than 15 percent of students did not abstract the problem enough to identify sub-problems for which methods could be written. This is in line with previous results [5], which state that some students find it hard to abstract problems enough to solve them computationally.

An often used metric [2, 3] is the amount of comments included in the program. Two out of the three expert programmers in this study used comments even in a simple task, whereas only about one in three students included any comments in their code. However, the quality of student's comments was not thoroughly analyzed and as such could merely contain leftover commented-out code or 'real' comments which explain what is happening or why. The experts' comments, however, were analyzed and one expert gave comments with informational value whereas the other expert merely commented out a method.

It seems clear that merely examining the surface elements of any produced code is not enough to make a distinction between the novice and the expert. Therefore a qualitative approach is applied: analyze the submission-chains of several students and compare them with the analysis of the experts' submission-chains. Two students' submission-chains were randomly selected for a qualitative analysis of the problem solving process which is then compared to that of the experts. The reason these two students were selected is as follows: one of the selected students had a submission count in line with that of the experts (6) and the other had a very high submission count (100). This distinction was made because the submission count seems to be the most prominent distinguishing statistical factor between experts and novices (table 1).

The next chapter will give a brief overview of the expert's submission-chains and then proceed to analyze the selected students' submission-chains by first dividing them into smaller parts, each of which will contain a particular approach. An explanation of the issue the programmer seems to be facing is also described based on investigating the lines of code the programmer has focused on between submissions. Then, the approaches taken by the students will be compared with those taken by the expert programmers in an attempt to discern qualities found in professional programmers, but not in programming students.

4 RESULTS

4.1 Submissions by experts

Three professional programmers participated in the study and solved a relatively simple algorithmic problem on the ViLLE learning platform using the Java programming language. The specific task can be seen in Figure 1. One programmer solved the problem in one submission, whereas the other two took 6 and 7 submissions to complete the same task.

The approach taken by Expert A, who solved the assignment in one submission, was to collect all the required crane positions and box manipulation commands into an array, then loop over that array and merely move the crane to the given position or execute the command. This approach has two benefits: simplicity and maintainability. The solution is simple and avoids any index off-by-one errors, which are notoriously common in the software industry. Secondly but also is very maintainable, since when the location of the boxes are changed, only the array containing the commands needs to be redone.

The other two expert programmers took a different approach: they created a method which flipped the order of the boxes by moving them one square ahead, then called the method twice. Expert B created the method for the very first submission, whereas Expert C first implemented the logic and created the method only when duplicate code would have otherwise been created for the second flip. For both these experts, most of the submissions were done in order to adjust the loop indexes to suit this particular arrangement of boxes. This approach produces more self-documenting code and the purpose of the code is more readily seen from it. However, it is not as general as the previous solution, since when the situation changes, everything needs to be recoded.

4.2 Submissions by students

Two students' submissions were selected to be analyzed. One student made 6 submissions, which is in line with the number of submissions made by the expert programmers, and the other student made 100 submissions. Only code that compiled and ran successfully was considered a submission. First Student A's submission-chain containing six submissions is analyzed and the approach is compared to that of the experts. Then, Student B's submission-chain of 100 submissions is given the same treatment. Both students ultimately arrived at a similar solution, which is described in Code snippet 1.

```
up(5)
right(5)
grab()
right(1)
down(3)
drop()
//continue in similar fashion until all boxes are moved

PROCEDURE up(amount)
    for(var i = 0; i < amount; i++)
        crane.up()
END

PROCEDURE down(amount)
    for(var i = 0: i < amount; i++)
        crane.down()
END
//similar prodecures for all directions
```

Code Snippet 1. Student solution

4.2.1 Student A

Student A begun solving the task using brute force: No methods were included, but multiple for-loops with equal content (for moving the crane) were used. The second submission introduces methods as shown in Code Snippet 1. The student clearly realized the amount of duplicate code of repeatedly moving the crane a certain amount of squares to the same direction and created a method in order to not have as much duplicate code. While the process bears resemblances to Expert C's process, one major difference here is the level of abstraction. The expert created a method to solve a subproblem for the task, whereas the student only wrapped a for-loop in a method call, presumably for ease of writing or convenience. Submissions 3 and 4 were equal; no code was changed. Presumably the program visualization was studied to determine the next step. The remaining submissions consisted of adding individual commands and correcting method parameters to move the crane appropriately. These submissions are in line with those of experts B and C, who also required several submissions at the end to fix loop indexes.

The final result of Student A resembles that of Expert A, in that both focus on executing commands for the crane. Both solutions are equally general, as the move methods can be reused for a different exercise with a different situation. However, the lack of using an array to store the commands makes the student's solution somewhat hard to follow, as it is rather verbose and long. Additionally, the relative positions for the crane required by this approach were apparently quite difficult to calculate, as they required several submissions to get right. This is contrast with the absolute positions used by the expert, which only required one submission. The final submission for Expert A was 51 lines of code, whereas Student A had a final submission with 77 lines of code. This is in line with the data in Table 1, where students, on average, write longer code than experts. This is, arguably, because experts have more experience with abstracting tasks. Thus, we can argue that strong abstraction skills lead to short code and more expert-like programming behavior.

4.2.2 Student B

Student B begun solving the task using brute force, much like Student A. The first submission to correctly organize the boxes is submission 57, which only received a 50% mark due to excessive crane movement. However, further submissions were made in an attempt to receive the maximum points. The final submission is almost identical to that of Student A, but the process in which it was achieved differs drastically. Student B submitted working code for the task 100 times, which is almost 17 times the average submissions made by the experts. Student B starts over three times, at submissions 19, 30 and 88.

The first approach taken by Student B, in submissions 1 to 19, was to spread the boxes horizontally on the ground, and then regroup them in the correct order. The code contained only for loops, some of which were nested. Presumably, the student started over either because the complexity of the code grew too high or a better idea was come up with. Either way, the next approach in submissions 19 to 30 was the same one Student A and all experts used: flip the order of boxes by moving them one square ahead. While the algorithm was the same, the implementation was not: the code consisted of two for loops to move the crane above the boxes, then commands to move the crane one square at a time. This approach was then dismissed after 11 submissions probably because the student realized the maximum score was unreachable due to the tremendous number of lines of code of a complete solution.

The submissions 30 to 57 retake the very first approach of lining the boxes horizontally, then regrouping them in the correct order vertically. The task is completed adequately during this phase in submission 57, but due to excessive crane movement, only half the possible points are awarded.

Due to the excessive crane movements previously, submissions 58 to 88 retake the second approach attempted in submissions 19 to 30, which have less crane movement. However, as before, the nested for-loops cause the complexity to rise too high for this student, indicated by a streak of 11 submissions in which only loop indexes are tweaked.

Submissions 88 to 100 continue the same approach as previously, but this time the complexity is reduced by having introduced the methods to move the crane several steps at a time. This obviously helped the student by easing some of the cognitive load; the problem is solved in only 12 submissions.

Student B seems to struggle with cognitive overload. The majority of submissions hint at forgetting the state which the crane and boxes are in. Most of the submissions seemed to be for the sole purpose of checking the state in order to be able to proceed. Remembering the location of the crane and all boxes at all times is extremely hard for people with a normal working memory. However, assuming the students abstract the state so that they only keep the location of the crane and the box they’re moving in mind, remembering the locations of these two items is not difficult. Something else must be using the cognitive resources of this student. One possibility is that the syntax of Java has not been internalized, so it, too, takes up valuable cognitive resources. This is especially noticeable in the submissions before the 88th, wherein the methods to move the crane a certain amount were introduced. After the 88th submission, the resources required to remember the syntax, and possibly function, of a for-loop was lifted, so more resources could be allocated to keeping track of the state of the crane and current box. After the introduction of the movement methods, the task was solved in only 12 submissions.

5 CONCLUSION

The task was relatively simple, which possibly caused the expert and student processes and final solutions to be similar. Perhaps a more fruitful study of this kind could be made with a more complex programming task. The processes of students and expert programmers will most likely differ drastically in tasks involving more Object-Oriented aspects, wherein software architecture needs to be taken into account.

The analysis of Student B's submissions revealed, that some students might have kept on submitting even after completing the task satisfactorily. This, then, means that the true number of unique submissions might be slightly less than reported. However, the average number of submissions by the students is unlikely to reach that of the experts, even if all surplus submissions are removed.

As stated in [3], not many static qualities of code differed not only between students, but also between students and experts. Instead, several key differences were found in the process of arriving to that solution. The main difference was that experts abstracted the problem more efficiently and earlier than students, which made it easy to construct the solution using these sub-solutions.

Experts seem to abstract the task at hand more than students, which is most likely due to existing templates, or plans, on how to solve a certain type of problem, as described in [8]. These templates should probably be taught to students. Further, students would benefit from instruction and strategies in how to abstract tasks.

REFERENCES

[1] A. Evans, W. Ferguson, J. Hartman and J. Neider, "Optimizing Program Performance," in *MIPS Compiling and Performance Tuning Guide*, 1994, pp. 463-465.

[2] B. Shneiderman, "Measuring computer program quality and comprehension," *International Journal of Man-Machine Studies,* vol. 9, no. 4, pp. 465-478, 1977.

[3] D. M. Breuker, J. Derriks and J. Brunekreef, "Measuring static quality of student code," in ITiCSE '11 Proceedings of the 16th annual joint conference on Innovation and technology in computer science education, 2011.

[4] D. Hagan and S. Markham, "Does it help to have some programming experience before beginning a computing degree program?," in *ITiCSE '00 Proceedings of the 5th annual SIGCSE/SIGCUE ITiCSEconference on Innovation and technology in computer science education*, 2000.

[5] D. M. Kurland, R. D. Pea, C. Clement and R. Mawby, "A study of the development of Programming ability and thinking skills in high school students," *Educational computing research,* vol. 2, pp. 429-458, 1986.

[6] E. Lahtinen, "A Categorization of Novice Programmers:A Cluster Analysis Study," in *Psychology of Programming Interest Group*, 2007.

[7] M. T. Chi, M. Barook, M. W. Lewis, P. Reimann and R. Glaser, "Self-explanations: How students study and use examples in learning to solve problems", *Cognitive Science,* vol. 13, no. 2, pp. 145-182, 1989.

[8] M. C. Linn and M. J. Clancy, "The case for case studies of programming problems," *Communications of the ACM,* pp. 121-132, March 1992.

[9] M. McCracken, Y. B.-D. Kolikant, V. Almstrum, C. Laxer, D. Diaz, L. Thomas, M. Guzdial, I. Utting, D. Hagan and T. Wilusz, "A multi-national, multi-institutional study of assessment of programming skills of first-year CS students," *ITiCSE 2001,* 2001.

[10] M. P. Uysal, "Improving First Computer Programming Experiences: The Case of Adapting a Web-Supported and Well- Structured problem-Solving Method to a Traditional Course," *Contemporary Educational Technology,* vol. 5, no. 3, pp. 198-217, 2014.

[11] R. Lister, E. S. Adams, S. Fitzgerald, W. Fone, J. Hamer, R. M. Morten Lindholm, J. E. Moström, K. Sanders, O. Seppälä, B. Simon and L. Thomas, "A Multi-National Study of Reading and Tracing Skills in Novice Programmers," *SIGCSE Bulletin,* vol. 36, no. 4, pp. 119-150, 2004.

[12] R. D. Pea, "Language-independent conceptual ”bugs” in novice programming," *Educational computing research,* vol. 2, no. 1, pp. 25-36, 1986.

[13] S. Bergin and R. Reilly, "Programming: factors that influence success," in SIGCSE '05 Proceedings of the 36th SIGCSE technical symposium on Computer science education, 2005.

[14] T. Gilb, *Software metrics,* Winthrop Publishers, 1977.

[15] T. J. McGill and S. E. Volet, "A Conceptual Framework for Analyzing Students' Knowledge of Programming," *Journal of research on Computing in Education,* vol. 29, no. 3, pp. 276-297, 1997.