PPIG'07 Full paper

A Categorization of Novice Programmers: A Cluster Analysis Study

Essi Lahtinen

Tampere University of Technology
Institute of Software Systems
Tampere, Finland
essi.lahtinen@tut.fi

Abstract. Beginning software engineering students often lack skills necessary to perform efficient programming work, such as comprehending or debugging program code. To facilitate the needs of students having different skill levels, teachers should be able to recognize the specific student groups.
Bloom's Taxonomy defines educational objectives for the development of students' cognitive skills. This paper presents a study of 254 undergraduate students of a basic programming course whose performance were measured on different levels of Bloom's Taxonomy. The results of statistical cluster analysis suggest that the obtained student groups did not align with the Bloom's Taxonomy: students performing poorly on lower levels can still perform well on higher taxonomy levels. Based on the results, this paper suggests six student groups the teacher should recognize when organizing basic programming courses.

1 Introduction

Programming is a versatile skill, requiring knowledge of programming languages and environments, as well as creative problem solving. It demands mastering numerous schemas, selecting the most suitable schemas for the problem to be solved, adapting and merging the schemas so that the final solution will be a new working combination of schemas [8].

Teachers often try to facilitate learning to write programs by making students read program code, which is a little contradictory [5]. In addition, students are given programming assignments as a means of gaining practical experience essential for developing their programming skills. However, Lister et al. [5] conclude that many students lack skills that are mandatory for problem-solving. The reason for this might be that teachers do not reckon with all kinds of learning difficulties in their teaching methods.

The aim of this work is to identify student groups that have problems with certain skills. Blooms Taxonomy [3] is used as the basis of this study since it is obvious that a professional programmer should be able to work on all its levels. Students were evaluated using simple assignments at different levels of the taxonomy and statistically analyzing the results to form groups of students with similar skill profiles. This grouping enables differentiating teaching efforts for student groups to meet their particular needs.

This study covers students' skills on different fields related to programming, for example, the ability to understand the meaning behind a code fragment. The intention is to take into account the possibility that the skills are not learnt in a respective order. We want to investigate if it is obvious that a group of students know the "easy" things and another group knows both the "easy" and the "difficult" ones. Can it be possible that there are students who only know the "difficult" things? Thus, we wanted to group the students using many different variables and chose the approach to form clusters of students by a statistical cluster analysis.

The next section introduces other studies where novice programmers are grouped and clarifies the idea behind this study compared to the others. Section 3 introduces the test settings and Section 4 presents the identified categories. Section 5 discusses the results further and the conclusions are presented in Section 6.

2 Related Research

Comprehension and generation of program code are examples of two different skills that a programmer needs. Being able to follow ready given solutions does not mean that you are able to write your own solutions. Surprisingly, the ability to write programs does not imply the ability to read another programmers code [10, 5].

In some studies, the initial setting is that students are grouped to successful and unsuccessful students according to their score in a test. For instance, Lister et. al [6] studied novice programmers' ability to understand and evaluate code fragments. They grouped students in quartiles and their result is that the students who belong in the highest two quartiles understand the meaning behind a code fragment remarkably better than the other students.

In addition to quantitative methods, it is possible to group students by qualitative methods. Perkins et al. [9] conducted a study simply by observing programming students and intervening the programming when needed by a couple of questions. They identified two ways of acting in a problematic programming situations and named the student groups *the stoppers* and *the movers*. Stoppers tend to give up whenever a problem occurs, and movers keep trying different kinds of solutions to proceed in the situation.

Phenomenography and variation theory are qualitative research methods used increasingly in computer science education research for identifying phenomena and different classes inside the phenomena. Eckerdahl and Berglund [4] studied students' opinions on what does it mean to learn programming by a set of semistructured interviews using phenomenography. Their result is a set of five inclusive categories starting from students thinking that programming is knowing a programming language and ending to students who see programming also as a way of thinking and problem solving that can be used outside the programming course too. Berglund and Wiggberg [2] found a similar inclusive categorization of the ways in which students act to learn computer science. These categories found phenomenographically often concern only one topic. In addition, it is labourious to analyze a large group of students phenomenographically.

On the field of programming, there are no earlier studies grouping students by cluster analysis. Cluster analysis has, however, been applied for studying engineering students' attitudes, orientations, motivations and intentions towards mathematics in Tampere University of Technology [7]. They discovered groups that they named surface oriented learners, peer learners, students needing support, independent learners and skilful students. This grouping has also been used in developing the mathematics teaching in TUT.

3 Experiment

In this experiment, students had to work out the answers to exercises which measured their competence in programming on different levels of Bloom's Taxonomy. The topic of the experiment was arrays. The exercise on the first level of Bloom's Taxonomy (*knowledge*) was simply to describe what kind of an array is declared in the example. The exercise on the second level (*comprehension*) was borrowed from the study by Lister et al. [6], since it has already proved to be a good way of measuring the students understanding of program code. In the exercise, the students had to explain briefly in simple language what a piece of code does with the array. In the third level exercise (*application*), the students were told to make a slight modification to the piece of code given earlier. On the fourth level (*analysis*), there was a question conserning the reasoning of the implementation of the loop structure in the code of exercise 2. On the fifth level (*synthesis*), the task was to write a piece of code from a scratch. However, since all the exercise dealed with arrays, the code of Exercises 1 and 2 contained all the syntax required for this exercise. The task was to build a new algorithm and write it in C++, not to memorize C++ syntax. On the highest level (*evaluation*), the students had to describe another algorithm for the same problem as in exercise 5 and compare the two solutions. The reasoning was also required. The questions can be seen in Appendix A.

The students taking part in the experiment were the participants of two introductory programming courses in TUT during the academic year 2005-2006. Most of them were first year undergraduates. The students had heterogenous backgrounds: some had earlier programming experience, some did not. The student group also included both computer science students and students from other technical faculties, e.g. material technology and mechanical engineering. The programming language used on the programming course was C++, and the students had to complete four or five small programming assignments during the course. The biggest programming assignment was approximately 500 lines of code.

The experiment was done in normal classroom setting using pen and paper to prevent the students from compiling and runing their code. They submitted the

Cluster number	1	2	3	4	5	6
Knowledge	0,37	-2,00	-2,65	0,41	0,41	0,14
Comprehension	0,47	0,50	-0,94	0,53	-1,10	-0,63
Application	0,48	-0,15	-1,35	-1,06	-0,95	0,48
Analysis	0,48	-0,10	-0,96	0,41	-0,34	-0,93
Synthesis	0,69	0,33	-1,41	-0,95	-0,71	-0,49
Evaluation	0,32	-0,68	-0,55	0,32	-1,06	0,27
Cluster name	Competent students	Practical students	Unprepared students	Theoretical students	Memorizing students	Indifferent students

Table 1. The final cluster centers of the K-means cluster analysis. The best grades are marked with light gray and the worst with dark gray.

answers signed, so the results can be compared to other data gathered during the programming course (exercise session attendance and homework programming assignment grades).

4 Results

Altogether 254 students took part in the experiment. The counts of students from the two different courses are 223 and 31. Additional information is not available from the course with the smaller student group.

The students' answers were graded on different scales depending on the degree of difficulty of the assignment. For instance, in the first exercise all the students had either completely correct answers or there were one or two details missing from the description. The solutions of the programming exercise (level 5, synthesis) varied greatly from no solution at all to a completely correct solution. In the grading, indexing out of bounds in an otherwise correct algorithm was graded as a smaller mistake than a completely misbehaving algorithm.

After grading the solutions, the grades were standardized statistically so that the mean value of each variable was 0 and the standard deviation 1. The students were grouped on the grounds of the standardized values using K-means cluster analysis.

In the following, we introduce a solution of six clusters that was identified the most informative grouping of the students by programming teachers. The final cluster centers are presented in Table 1. The table contains the standardized mean values of all the variables in the columns marked by the cluster number.

The cluster analysis also included the ANOVA test for the standardized values. The Sig-values for all the variables showed significant differences between the cluster centers ($p < 0.001$). However, the ANOVA test can only be used for descriptive purposes, since the whole idea of clustering is to maximize the differences between the groups. Thus, we can use this test only to say that each of the variables makes a difference at least between two of the cluster centers.

The positions of the cluster centers presented in Table 1 give a general view on the differences of the student groups. In addition, the students' exercise answers were studied again cluster by cluster to recognize common patterns. The clusters were named according to this information and they are introduced in the following subsections. The names are also presented in Table 1.

4.1 Group 1: Competent students (n=119)

The exam was not difficult for this—the largest—group of students. They had learned all aspects of programming fairly. These students had their biggest difficulties on the level evaluation, which was expected since it is often difficult for novices to reach the higher cognitive skills [3].

4.2 Group 2: Practical students (n=20)

This group of students had succeeded best in comprehension and synthesis, average in application and analysis, and their weekest skills were knowledge and evaluation.

Since this group can write program code, it is surprising that the result of the knowledge question was unsatisfactory. It seems that array was such a familiar concept to these students that they did not recognize to mention that there can be different kinds of arrays. Most of these students forgot to mention that the type of the element in the defined array is integer and some forgot that the size is also defined. Some students in this group also used very inaccurate terminology in the knowlege question. One could say that even if these students can write programs they are not interested in studying programming and thus they did not learn the terminology accurately.

In the programming assignment (synthesis), this was the only group where all the students produced an algorithm that was almost or completely correct. There were no extra `else` blocks or nested loops that were the most common problems with the algorithm in the other groups of students. The only problems this group faced were the loop structure looping once too little or once too much. I.e., an element was left out of the checking or there was an index out of bounds error. The basic idea of the algorithm was always correct though.

Writing program code did not seem to be a problem for this group since a few students had written their alternative idea for the algorithm in the evaluation exercise also as C++ code. Maybe explaining the idea in words was more difficult for these students. These students should concentrate on the big picture and practise also the evaluation and analysis skills.

4.3 Group 3: Unprepared students (n=12)

This group of students did not seem to prepare for the exam at all. They had not succeeded in any of the exercises. Since the programming homework assignments that were completed individually before the exam were much more difficult than the exam, one can rise a question about plagiarism.

4.4 Group 4: Theoretical students (n=26)

These students were in a way opposite to the practical students. They did well on levels knowledge, comprehension, analysis, and evaluation and had problems in application and synthesis, which require modifying or writing program code. They have learnt to read program code but have difficulties in producing programs on their own.

One could say that these students seem to think on a higher abstraction level. In the assignments where they had to work with C++, they had plenty of careless mistakes. For example, indexing out of bounds was very common. Also Winslow [10] has identified students who can solve problems by hand but have difficulties in expressing their solution in a programming language.

In the evaluation assignment they had explained and analyzed algorithms clearly, e.g., "You can also check if the content of the array is a palindrome by comparing the first and the last, the second and the secondlast value of the array. You can do these comparisons until the end or if you want to save time you can stop when you reach the middle of the array". In the synthesis part of the exercise when trying to implement this in C++, 27% of the students in this group had tried to use two nested loop structures to implement the described algorithm. The verbal presentation showed that the students had learnt a lot about programming—at least the concept of algorithm was clear—but the implementation caused difficulties. It was also rather common that these students put an `else` block after the `if` even if that made the algorithm wrong.

4.5 Group 5: Memorizing students (n=34)

Knowledge was the only level where this student group had done better than average. All the other skills caused problems. They had lerned by heart what is an array and almost all the students mentioned the size of the array and the type of the element that, for instance, the group 2 often forgot. However, this group of students lacked the interpretation of their knowledge. The same kinds of problems have been reported in other studies too. For example, Winslow [10] states that novices know programming concepts but fail to apply them when creating their own programs.

In the analysis exercise, many of the students in all groups had just simply explained that the index of the first element of an array is 0. This is true but the answer to the question requires deeper analysis of the program code. In this group the students had explained the consequences of the index numbers beginning from 0 very carefully and in detail. Many of the students had drawn a picture that was presented in the lecture material. However, the deeper analysis of this specific program code was not done.

In the exercise where they had to write a piece of code on their own, many students in this group had tried to make the problem simpler by handling only arrays of one certain size. For instance, you can check whether an array of four elements is a palindrome by only one `if`-structure like this:

Cluster Number	Mean	N	Std. Deviation
1	2,45	108	0,73
2	2,10	10	0,74
3	2,11	9	1,05
4	2,29	21	0,72
5	1,96	27	0,94
6	2,32	38	0,77
Total	2,32	213	0,79

Table 2. The mean values of students' programming assignment grades by their cluster numbers.

```
if( array[ 0 ] == array[ 3 ] && array[ 1 ] == array[ 2 ] )
```

None of the students in the other groups had tried this kind of simplifications.

A typical answer for the evaluation exercise in this group was "My solution is good but certainly there are also other good ways of doing the same". The teachers of the programming courses had often reminded that there is always more than one solution to every problem. This group of students was often not able to come up with an idea for the other solution.

4.6 Group 6: Indifferent students (n=43)

This group of students was the most difficult to name. Since they seem to lack interest on the topic, they were named "indifferent". In most of the skills they were close to average. They had succeeded well in the application exercise and their second best skill was evaluation where they had to think about modifications to the algorithm. However, they had difficulties in analyzing and understanding the behaviour of the program code (analysis and comprehension) and writing code on their own (synthesis).

It appears strange that students who do not understand the meaning of the piece of code well are still able to modify the code. In this case the modification did not require understanding the deeper meaning of the program code if you recognize that you can only change the loop structure.

This group of students is capable of doing little modifications to program code but writing program code on their own seems to be difficult.

4.7 Analysis of the student groups

The mean values of students' programming assignment grades presented in Table 2 support some of the analysis of the group. The only statistically significant difference ($p < 0.05$) of the assignment grades is between the competent students (group 1) and the memorizing students (group 5). Still we can recognize that the practical students (group 2) have not received too good grades even if they succeeded to write correct code in the test situation. The reason for this is certainly that the grading of the programming assignments did not handle only program correctness but also design and style. Program design requires evaluation skill that the practical students should improve.

It is also noteworthy that the unprepared students (group 3) have fair grades of their homework programming assignments. Since they have practised programming it is exceptional that they performed so badly in the test. Plagiarism could be one of the explanations for this. One can also question if the experiment measured the skills learned in the homework assignments.

Theoretical students (group 4) and indifferent students (group 6) have done well in their programming assignments. They could certainly be part of the top students with a little extra rehearsal.

5 Discussion and Future Work

Bloom's Taxonomy has a basic chronological element: to reach the higher cognitive levels the student is to proceed linearly through the hierarchy, reaching the lower levels first. There has been discussion on the order of the levels. For example, Anderson and Kratwohl [1] have suggested that the top two or three levels of the taxonomy may be parallel.

The categorization presented in this paper utilizes the conceptual levels of Bloom's Taxonomy but does not follow such a chronological order. The study seems to suggest that students can perform well at higher levels even when they have problems at lower levels of the taxonomy. As an example, the theoretical students (group 4) performed well on level 6 while having problems with levels 3 and 5. The results revealed interesting groups of students whose needs could be taken into account by carefully targeting teaching and extra assignments.

Since the experiment did not completely correspond to Bloom's Taxonomy, it is important to assess the threats to internal and external validity of the study. Bloom's Taxonomy is not rigorously defined but deliberately left open for interpretation. It is possible to argue whether the questions have been properly assigned to different levels of Bloom's Taxonomy. For instance, in contrast to professional programmers, beginning students could perceive the comprehension level question used in this study already as an analysis level question.

The study was performed using one set of test questions only. It is therefore possible that some students failed to give right answers due to misinterpreting or misunderstanding the questions, and not lack of skills. Repeating the study with different set of questions and with a bigger population of students would offer additional insight to the presented grouping of students. Another potential future line of study is using a different categorization scheme for grouping the students, for instance, another taxonomy.

6 Conclusions

Between the competent students and the unprepared students there are four other student groups who have their own kind of problems with different skills related to programming. The practical students and the theoretical students have an opposite perspective to programming. The others are able to do programming and the others are able to analyze programs. The memorizing students know programming concepts but have problems on all the other fields. The last group, indifferent students, should be studied better to understand their problems deeper.

It is often difficult for a seasoned professional to appreciate the kinds of difficulties a novice programmer can encounter. Investigating and analyzing the students' problems can help the teachers to gain a better insight into these difficulties. The study presented in this paper is expected to lay ground for future work for finding solutions to be used in classrooms.

7 Acknowledgements

Nokia Foundation has partly funded this work. I wish to thank Miika Huikkola from TUT Institute of Mathematics for his support with the statistical analysis and Petri Selonen from TUT Institute of Software Systems for his useful comments on my work.

References

1. L. W. Anderson and D. A. Kratwohl. *A Taxonomy for Learning, Teaching and Assessing: A Revision of Bloom's Taxonomy of Educational Objectives*. Addison-Wesley, 2001.
2. A. Berglund and M. Wiggberg. Students learn cs in different ways: insights from an empirical study. In *ITICSE '06: Proceedings of the 11th annual SIGCSE conference on Innovation and technology in computer science education*, pages 265–269, New York, NY, USA, 2006. ACM Press.
3. B. S. Bloom, M. D. Engelhart, E. J. Furst, W. H. Hill, and D. Krahwohl. *Taxonomy of Educational Objectives, Handbook I: Cognitive Domain*. David McKay Company, Inc., New York, 1956.
4. A. Eckerdal, M. Thune, and A. Berglund. What does it take to learn 'programming thinking'? In *ICER '05: Proceedings of the 2005 international workshop on Computing education research*, pages 135–142, New York, NY, USA, 2005. ACM Press.
5. R. Lister, E. S. Adams, S. Fitzgerald, W. Fone, J. Hamer, M. Lindholm, R. McCartney, J. E. Mostrm, K. Sanders, O. Seppl, B. Simon, and L. Thomas. A multinational study of reading and tracing skills in novice programmers. In *ITiCSE-WGR '04: Working group reports from ITiCSE on Innovation and technology in computer science education*, pages 119–150, New York, NY, USA, 2004. ACM Press.
6. R. Lister, B. Simon, E. Thompson, J. L. Whalley, and C. Prasad. Not seeing the forest for the trees: novice programmers and the solo taxonomy. In *ITICSE '06: Proceedings of the 11th annual SIGCSE conference on Innovation and technology in computer science education*, pages 118–122, New York, NY, USA, 2006. ACM Press.

7. S. Pohjolainen, H. Raassina, K. Silius, M. Huikola, and E. Turunen. Clustering of students of engineering mathematics based on their attitudes, orientations, motivations and intentions. In *Proceedings of the 4th WSEAS / IASME International Conference on Engineering Education*, 2007.
8. A. Robins, J. Rountree, and N. Rountree. Learning and teaching programming: A review and discussion. *Computer Science Education*, 13(2):137–172, 2003.
9. E. Soloway and J. Spohrer. *Studying the Novice Programmer*. Lawrence Erlbaum Associates, Hillsdale, New Jersey, 1989.
10. L. E. Winslow. Programming pedagogy – a psychological overview. *SIGCSE Bulletin*, 28(3), September 1996.

Appendix A

The exercise questions:

1. What kind of a variable is defined here? `int values[ SIZE ];`
2. Let's assume that the required `include`-directives, the declarations of the variables etc. are in order. Explain briefly in simple language what does the following piece of code do.

```
bool test = true;
for( unsigned int i = 0; i < SIZE - 1; ++i ) {
    if( values[ i ] > values[ i + 1 ] ) {
        test = false;
    }
}
```

3. Modify the piece of code in assignment 2 so that it will go through `values` in reverse order but the functionality will be exactly the same.
4. Why does the condition of the `for`-loop in the piece of code in assignment 2 compare the variable `i` to the value `SIZE - 1` and not the value `SIZE`?
5. Write a piece of code that will check if a sequence of numbers is the same from beginning to end and from end to beginning. For instance, 1, 2, 1 and 9, 7, 7, 9 are but 6, 7, 9, 6 is not.
6. Evaluate the solution you wrote to exercise 5. Is it possible to implement the same functionality with another algorithm? Would the other algorithm be better or worse than the algorithm you implemented? State reasons for your answer.