122nd ASEE Annual Conference & Exposition
June 14 - 17, 2015
Seattle, WA
Paper ID #11804

Quantifying Student Progress through Bloom's Taxonomy Cognitive Categories in Computer Programming Courses

Dr. Candido Cabo, New York City College of Technology/City University of New York

Candido Cabo is a Professor in the Department of Computer Systems Technology at New York City College of Technology, City University of New York (CUNY). He earned the degree of Ingeniero Superior de Telecomunicacion from the Universidad Politecnica de Madrid (Spain) in 1982, and a Ph.D. in Biomedical Engineering from Duke University (Durham, NC) in 1992. He was a post-doctoral fellow at Upstate Medical Center, State University of New York (Syracuse, NY), and a research scientist in the Department of Pharmacology at the College of Physicians and Surgeons of Columbia University (New York, NY). Since 2005, he has been a member of the doctoral faculty at the CUNY Graduate Center. His research interests include computer science and engineering education and the use of computational models to understand and solve problems in biology.

Quantifying Student Progress through Bloom's Taxonomy Cognitive Categories in Computer Programming Courses

Abstract

Computer programming courses are gateway courses with low passing grades, which may result in student attrition and transfers out of engineering and computer science degrees. Progress in student learning can be conceptualized by the different cognitive levels or categories described in Bloom's taxonomy, which, from the lowest to the highest order processes, include: knowledge, comprehension, application, analysis, evaluation, and synthesis. The purpose of this study is to gain insight into how students transfer their conceptual knowledge and comprehension of computer programming concepts (knowledge and comprehension categories in Bloom's taxonomy) into their ability to write computer programs (application category in Bloom's taxonomy), using Bloom's taxonomy as a framework.

A total of 62 students who took a first computer programming course using Java participated in this study from spring 2013 to spring 2014. Novice computer programming students face two barriers in their progress to become proficient programmers: a good understanding of programming concepts (first two categories in Bloom's taxonomy) and the ability to apply those concepts (third category in Bloom's taxonomy) to write viable computer programs. About 35% of students had an acceptable performance in both conceptual understanding of programming concepts and ability to write viable programs. About 44% of students had an inadequate performance in both concepts and programming skills. 16% of students had an adequate understanding of computer concepts but were unable to transfer that understanding into writing viable computer programs. Finally, 5% of students were able to produce viable computer programs without an adequate conceptual understanding. Of the students who had adequate understanding of computer concepts, 69% were able to write viable computer programs. Linear regression modeling suggests that conceptual understanding is a good predictor ($r^2 = 74\%$) of the ability to apply that knowledge to write computer programs. Factor analysis identified two factors grouping the interdependencies and correlations between programming concept categories: the first factor included repetition and classes; the second factor included syntax, assignment, methods and arrays. Multiple regression analysis shows that a subset of conceptual assessments consisting of repetition, classes, assignment operations and Java syntax is sufficient to predict students' ability to write viable programs ($r^2 = 0.78$).

In conclusion: 1) Adequate average performance in programming concepts is *necessary* but *not sufficient* for students to write viable computer programs; 2) Adequate performance in *all* individual conceptual categories, and not just adequate *average* performance, is necessary to be able to write viable computer programs; 3) Given the correlations between performance in different conceptual categories, a subset of conceptual assessments consisting of repetition, classes, assignment operations and Java syntax is sufficient to predict students' ability to write viable programs; 4) Low values of r^2 may indicate that concepts taught and expected practical skills are not properly aligned. Thus regression analysis can be used to improve the alignment between concepts and skills facilitating student progress through the different cognitive levels in Bloom's taxonomy.

1. Introduction

Computer programming courses are gateway courses with low passing grades, which may result in student attrition and transfers out of engineering and computer science degrees. Progress in student learning can be conceptualized by the different cognitive levels or categories described in Bloom's taxonomy [1], which, from the lowest to the highest order processes, include: knowledge, comprehension, application, analysis, evaluation, and synthesis. Bloom's taxonomy is a conceptualization of learning objectives that helps educators assess student progress through different cognitive levels in a given discipline [1]. Krathwohl's [13] revision of Bloom's taxonomy can be applied to the learning of computer programming. It emphasizes remembering knowledge (can the learner recall or remember information, i.e. computer programming concepts?), comprehending that knowledge (can the learner explain programming concepts?), applying (can the learner use the concepts in a new way to solve problems, i.e. to write viable computer programs?), analyzing (can the learner distinguish among the different parts of a problem; can the learner use this skill to debug and troubleshoot a computer program?), evaluating (can the learner justify a stand, decision or solution to a given problem?), and, finally, creating (can the learner plan and generate a novel product, point of view or solution to a problem?).

In an earlier study [3] we found that there are two barriers for student success in computer programming courses: a good understanding of programming concepts and the ability to apply those concepts to write viable computer programs. Factor analysis showed that student understanding of computer programming concepts falls in two meta-conceptual groups: an "algorithmic" (repetition, selection and classes) and a "structural" (methods, arrays and assignment) factor. Moreover students had a better understanding of concepts that relate to the "algorithmic" factor than of concepts that relate to the "structural" factor. Student performance in the "structural" conceptual component was predictive of the student's ability to solve practical computer programming problems. To improve student performance we suggested that strong emphasis in the "structural" components of computer programming (assignment, methods and arrays) is necessary for a successful transition from concepts to skills in computer programming courses. Overall, that study suggested that student knowledge and comprehension in seemly independent computer programming concepts show hidden correlations and interdependencies, and that some programming concepts are more important than others in predicting students' ability to write viable computer programs than others.

The purpose of this study is to gain further insight, in a larger group of students, into how students transfer their knowledge and comprehension of computer programming concepts (knowledge and comprehension categories in Bloom's taxonomy) into their ability to write viable computer programs (application category in Bloom's taxonomy), using Bloom's taxonomy as a framework. The following research questions were addressed in this study: 1) Is adequate performance in conceptual understanding sufficient for a student to write viable computer programs? 2) How big is the gap between conceptual understanding of programming concepts and the ability to apply those concepts to write viable computer programs? 3) Are some concepts more important than others in determining students' ability to write viable programs?

2. Background and Related Work

A number of challenges faced by novice programmers have been identified over the years by the computer science education community. It has been shown that an understanding of the problem domain to be solved by implementing a computer program should be a prerequisite for writing the computer program itself [2, 6, 18, 29]. Students' inability to create a mental model [11] of a given problem domain hinders their ability to develop problem-solving skills and write computer programs.

Another difficulty encountered by novice programmers is the syntax of computer programming languages, which is often overwhelming to students who get distracted from solving problems by the obscurity of the statements and program organization. This difficulty was recognized early in computer programming education, and different strategies including graphical languages and animations of program states were developed [21]. One approach to increase success in first-year programming courses is a shift from teaching programming to teaching problem-solving skills [7, 8, 12]. This approach has been successful and avoids some of the problems that hinder progress in the development of thinking skills that are important for computer programming. However, this approach has also been criticized because the translation of a problem solution to a computer program is not obvious [18, 22]. The challenges faced by students and educators in learning and teaching computer programming have been summarized in a recent review [19].

Following earlier findings in computer education research we require our students to take a problem-solving course before their first programming course. It has been shown that introducing narrative elements in pre-programming problem-solving courses (a pedagogical approach that has been called programming narratives) is more effective than traditional approaches using a full-fledge programming language as a tool to help students develop computer programming problem-solving skills [4, 14, 15]. To facilitate the implementation of programming narratives we currently use *Alice* (www.alice.org), a programming environment that allows learners to create interactive animations while learning computer programming concepts. However, despite the benefit of using programming narratives to help students develop problem-solving skills, the transition from pre-programming problem-solving courses to courses where students should master a full-fledge programming language remains a challenge [18, 22].

3. Methods

3.1 Study Context and Participants

Our institution is one of the most racially, ethnically, and culturally diverse institutions of higher education in the northeast United States: 31% of our students are African American, 35.6% are Hispanic, 20.6% are Asian or Pacific Islanders, 11.6% are Caucasian, 0.5% are Native Americans, and 1.2% Other. The College's fall 2014 enrollment was 17,374.

We report data from performance assessments from 62 students who took a Programming Fundamentals course from spring 2013 to spring 2014. In this course, students use Java as the programming language of choice to help develop their conceptual and practical programming skills. For all students, this is the first programming course in their curriculum. However, before this course, all students had taken a problem-solving course in which they used pseudocode,

flowcharting and Alice (www.alice.org) to learn basic procedural and object-oriented programming concepts. The goal of the problem-solving course is to teach programming concepts without the burden of learning a full-fledge programing language. Basic Java programming is introduced in the last three weeks of the problem-solving course to facilitate the transition to the Programming Fundamentals course.

3.2 Exploratory Factor Analysis

Computer programming concepts assessments were grouped into seven different categories: assignment, repetition (for/while structures), selection (if/else structures), methods, arrays, classes and general syntax. Student performance in concepts and skills (i.e. ability to write viable computer programs) was assessed at three different times during the semester.

Exploratory factor analysis is a data reduction technique that aims at finding hidden correlations and interdependencies between different variables and grouping them in a number of overarching factors or components. Estimating the number of factors is tricky, and therefore to estimate the number of factors in the factor analysis we used different criteria. We used SPSS to extract the number of factors using the Kaiser-Guttman [9] (number of eigenvalues greater than one) and Cattell's scree test [5]. We also used FACTOR [16] to estimate the number of factors using Horn's parallel analysis [10]. The Kaiser-Meyer-Olkin measure of sampling adequacy was 0.674, above the suggested minimum of 0.5. Interpretation of the extracted factors can be made easier by orthogonal factor rotation. We used the varimax rotation method with Kaiser normalization.

3.3 Multiple Regression

To analyze the predictive value of student performance in computer programming concepts (the predictors or independent variables) on their ability to write viable computer programs (the dependent variable or the variable we want to predict) we performed multiple regression analysis using SPSS. Multiple regression analysis provides a model quantifying the relative contribution of each of the predictors towards explaining the total variance of the dependent variable. An independent variable coefficient was considered statistically significant and therefore contributing significantly to the prediction if $p < 0.05$.

4. Results

4.1 Student Performance in Computer Programming Concepts

Figure 1 shows average student performance in the seven types of programming concept assessments which were aimed at estimating student cognitive level in the first two categories of Bloom's taxonomy: knowledge and comprehension. Average performance ranges from about 8.2 in syntax related assessments to about 5.5 in assessments testing the understanding of the assignment operator. In all categories except for the use of repetition loops (for and while) and syntax, average performance is below 7, a level that could be used as marking the level of adequate performance (equivalent to a passing grade or C).

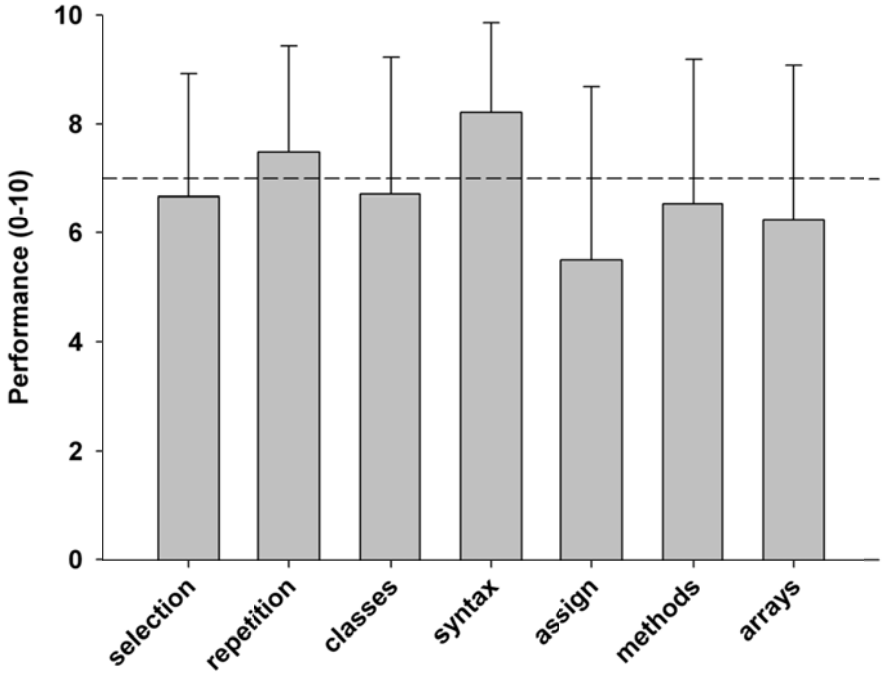

Figure 1. Average student performance (0-10) in seven types of programming concept assessments (first two categories in Bloom's taxonomy). n = 62. Dashed horizontal line marks a level of acceptable performance.

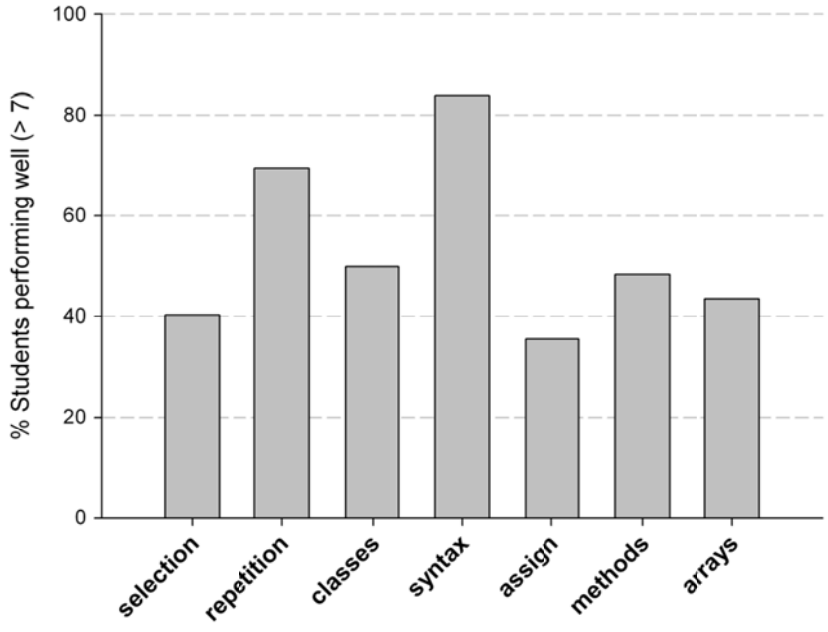

Figure 2. Percent of students performing adequately (>= 7 which is equivalent to a passing grade, 70% or C) in the seven different programming conceptual categories.

Average values can be distorted by very good performance of some students and/or very bad performance of others. Figure 2 shows the percent of students who performed well (i.e. who obtained >= 7 in the assessments) in the different programming concept assessments. The figure illustrates why computer programming courses are gateway courses with low passing grades: in most conceptual categories less than 50% of students performed at an adequate level.

4.2 Exploratory Factor Analysis

To further understand the nature of students' understanding of computer programming concepts, and the hidden correlations and interdependencies between programming concepts in those seven different categories, we performed exploratory factor analysis.

We grouped student performance in computer programming concepts assessments in seven different categories: assignment, repetition, selection, methods, arrays, classes and syntax. Factor analysis identified two factors or components grouping the interdependencies and correlations between programming concept categories. Figure 3 shows a plot of the factor loadings in the orthogonally rotated space, which illustrate the percent of correlation of the different conceptual categories with a given factor or component.

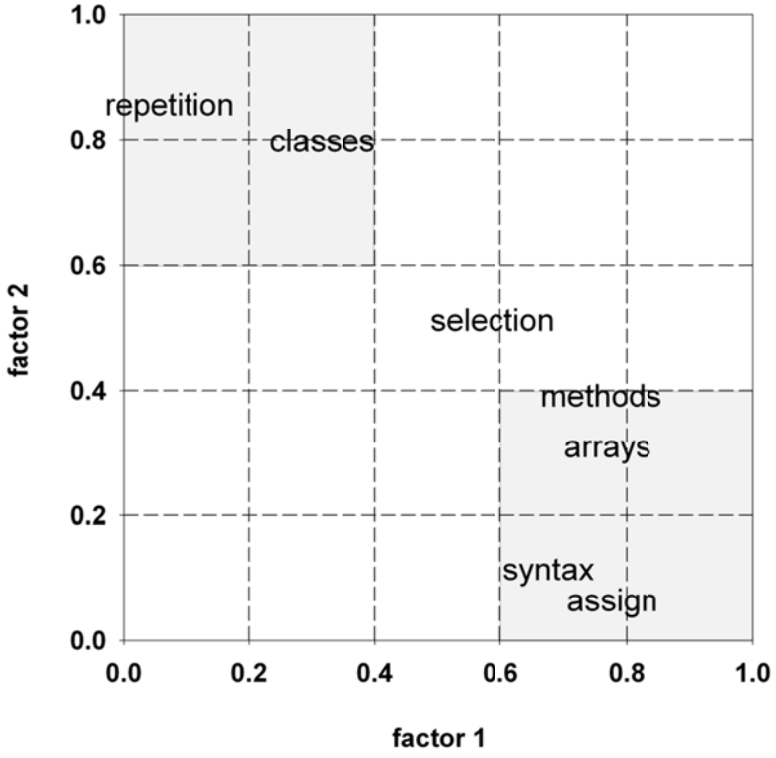

Figure 3. Factor plot illustrating the correlations of the seven conceptual categories with two factors. The categories repetition and classes are mostly correlated with factor 2 (correlation > 0.60). The categories methods, arrays, syntax, assignment are mostly correlated with factor 1 (correlation > 0.60). The correlation of the selection category is < 0.60 with either factor 1 or factor 2.

Factor loadings are the correlations between the different categories and the extracted factors (components), and therefore their value is between +1 and -1. The categories repetition and classes had a higher correlation with factor 2 (correlation > 0.60) than with factor 1 (correlation < 0.4). On the other hand, the categories methods, arrays, syntax and assignment had a higher correlation with factor 1 (> .60) than with factor 2 (< 0.40). The category selection was not strongly correlated with either factor 1 or 2. These results suggest that student performance in the different programming concept categories is not totally independent, and students tend to do well or poorly in groups of concept categories.

4.3 Relationship Between Performance in Programming Concepts and Skills

Students perform better in conceptual assessments (6.8±1.8) than in assessments testing their ability to apply those concepts to write viable computer programs (6.2±2.6; $p < 0.05$). Performance in concepts can be mapped to the first two levels of Bloom's taxonomy learning structure (knowledge and comprehension level), and performance in skills can be mapped to levels three and four (application and analysis level).

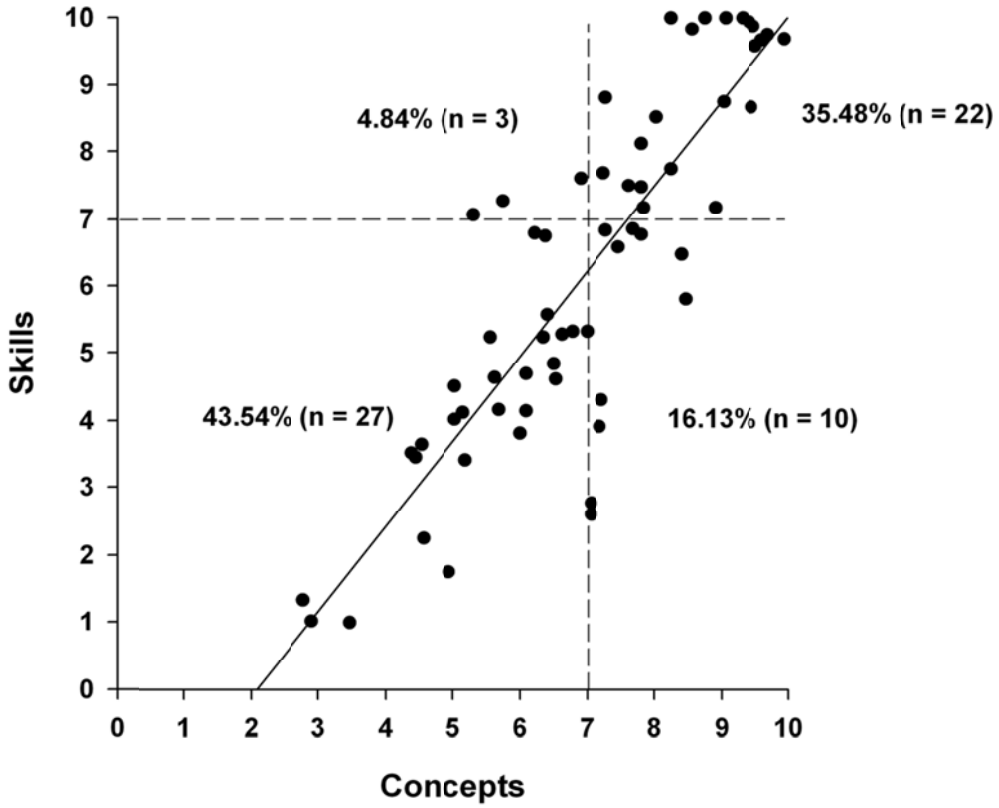

Figure 4. Average student performance in programming concepts and skills (n = 62). Dashed lines mark an acceptable performance in computer programming concepts and skills assessments (70%). The solid line is the regression line (Skills = 1.265 *Concepts – 2.64, with $r^2 = 0.74$).

Figure 4 shows the individual performance of students in concepts and skills assessments (range 0-10). As before, we considered 70% (equivalent to a C grade) an acceptable ("passing") performance in the assessments (vertical and horizontal dashed lines in Figure 4). About 35% of students had an acceptable performance in both conceptual understanding of programming concepts and ability to write viable programs. About 44% of students had an inadequate performance in both concepts and programming skills. 16% of students had an adequate understanding of computer concepts but were unable to transfer that understanding into writing viable computer programs. Finally, 5% of students were able to produce viable computer programs without an adequate conceptual understanding. Of the students who had adequate understanding of computer concepts, 69% were able to write viable computer programs.

The results in Figure 4 show that novice computer programming students face two barriers in their progress to become proficient programmers: a good understanding of programming concepts (first two categories in Bloom's taxonomy) and the ability to apply those concepts (third and fourth category in Bloom's taxonomy) to write viable computer programs.

Our analysis of Figure 4 relies in a somehow arbitrary threshold that marks the difference between satisfactory and poor performance in concepts and skills (70%, dashed lines in Figure 4). However, an analysis of the regression line (which is not dependent in arbitrary thresholds) leads us to similar conclusions. Linear regression (solid line in Figure 4, $r^2 = 0.74$) indicates that performance in programming concepts is highly correlated with performance in practical programming skills and that conceptual understanding is a good predictor of the ability to apply that knowledge to write computer programs.

Figure 4 shows that performance in concepts correlates with performance in skills. Only ~5% of students had an acceptable performance in skills but poor performance in concepts. This suggests that without a good grounding in the understanding of the concepts, it is very unlikely to develop acceptable practical programming skills. To further understand the role of understanding programming concepts in the ability to write viable computer programs, we compared performance in the different conceptual assessments in three different groups of students (Figure 4): students with adequate performance in concepts and skills (CS), students with adequate performance in concepts but not in skills (CNS), and students with poor performance in both concepts and skills (NCNS). The results are shown in Figure 5.

Comparison between the CS and CNS groups (Figure 5, white and light gray bars respectively) shows that their performance in the classes, syntax, methods and arrays categories is similar (< 10% difference); however, performance differences in selection, repetition and assignment are much larger (~15-20%). In fact average performance for the CNS group for categories selection, repetition and assignments is below or barely adequate (<= 7). Average performance for the CS group is well-above adequate (>= 8) for each individual conceptual category. This result seems to suggest that a minimum level of understanding of each individual concept (not just their average) is important to develop the ability to write viable computer programs. This is somehow expected because, in general, writing computer programs requires knowledge and comprehension of several programming concepts.

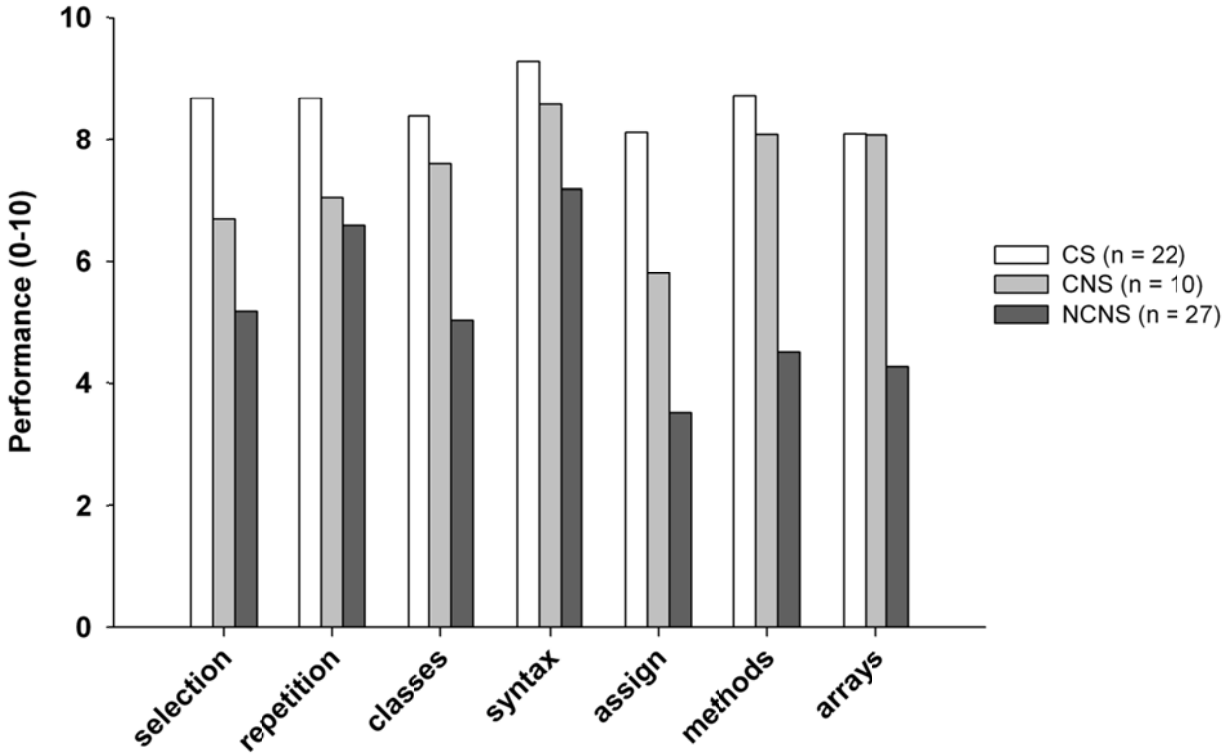

Figure 5. Average performance (range 0-10) in computer programming concepts assessments for: students with acceptable performance in both concepts and skills (CS, white bar); students with an acceptable performance in concepts but not in skills (CNS, light gray bar); students with poor performance in both concepts and skills (NCNS, dark gray bar).

Multiple regression analysis indicates that performance in programming concepts are predictive of the average performance in practical assessments ($F(7, 95) = 27.48$, $p < 0.05$ and $r^2 = 0.78$). Student performance in repetition, classes, Java syntax and assignment assessments contributed significantly to the prediction of performance in practical skills ($p < 0.05$). Performance in selection, methods or arrays assessments did not contribute significantly to the prediction. Therefore, the results show that performances in some concepts are better predictors of programming skills than others.

5. Discussion

We have analyzed progress in student computer programming learning using the cognitive levels or categories described in Bloom's taxonomy [1,13]. In the context of computer programming, the first two levels (knowledge and comprehension) can be quantified from performance in computer programming concepts, and the third and fourth levels (application and analysis) from the students' ability to write viable computer programs. We found that 44% of students are unable to reach an adequate level of conceptual knowledge and comprehension of computer programming

concepts, and, as a consequence, they cannot progress to write viable computer programs. 35% of students showed adequate performance in conceptual knowledge and were able to write viable computer programs. Multiple regression analysis shows that performance in conceptual knowledge is a good predictor of students' skills in practical assessments (78%). Therefore, adequate performance in programming concepts is necessary for students to write viable computer programs.

However, adequate average performance in programming concepts may not sufficient to write viable programs. 16% of the students cannot transfer conceptual knowledge and comprehension into viable computer programs (the application and analysis levels in Bloom's taxonomy), suggesting that other factors may also contribute to their ability to write computer programs. For example the ability to understand a problem, by being able to translate a word problem into an algorithm, is likely to contribute to students' performance in writing computer programs. Others have shown that an understanding of the problem domain to be solved is a prerequisite for writing a viable computer program [2, 6, 18, 29]. Even though the average of performance in programming concepts of students who cannot transfer to practical skills is adequate (Figure 4), in some individual categories their performance is below proficiency (selection and assignment, light gray bar in figure 5). This suggests that to be able to write computer programs, students will have to be proficient in several if not all conceptual categories.

Figure 4 shows that ~5% of the students (n = 3) are able to produce viable computer programs without an adequate understanding of concepts. A closer examination of Figure 4 indicates that the concept vs. skills performance of students in this group is close to the border with other performance regions (i.e. it is close to the dashed lines in Figure 4). For example, the performance of the two students with the lower performance in concepts (5.31 and 5.75) was marginally above adequacy in skills (between 7.1 and 7.3), which means that those two students could have been in the group of students with poor performance in concepts and skills. The third student, who had a higher performance in skills than the other two (7.6), performed just below adequacy in conceptual assessments (6.9), which means that this student could have been in the group of students with adequate performance in concepts and skills. So, it is possible that different assessments could have classified those students in different performance groups. An earlier study[4] on student performance on problem-solving also showed that 5-7% of students can have an adequate performance in problem-solving skills without an adequate performance in problem-solving concepts. Even though the small number of students in this group prevents us from extracting general conclusions, we have observed that some students lack metacognition skills and, as a consequence, they are not aware of their own knowledge. For example, some students may not be able to produce a valid answer to the question, "How would you declare a variable that could contain a number in Java?," but they may be able to declare and use a numeric variable in a viable Java computer program. The role of metacognition skills in students' understanding of programming concepts and in their ability to write viable computer programs deserves further study.

In an earlier study [3], we identified two factors grouping the interdependencies and correlations in student understanding of programming concept categories. The first component correlated with the repetition and selection categories, and could be referred to as the "algorithmic" component. The second component correlated with the methods, arrays and assignment categories, and could

be referred as the "structural" component. The results shown here for a larger student population are consistent with those earlier results. Those two factors may correspond to problem-solving skills and programming skills, which according to others [7, 8, 12] correspond to two different sets of cognitive skills.

Our students take a pre-programming course before they take their first programming course (see Methods/Participants) to develop problem-solving skills. Still, to be able to write viable computer programs, students need both problem-solving and programming skills. Despite the benefits of an approach teaching problem-solving skills first, the transition from pre-programming problem-solving courses to courses in which students should master a full-fledge programming language remains a challenge [18, 22]. This is reflected in the number of students (44%) who did not have an acceptable performance in either concepts or skills (Figures 4 and 5). Even though those students had passed a previous problem solving course, they find the transition to a learning environment that uses a full-fledge programming language like Java difficult.

According to Mayer [17], in addition to the cognitive and metacognitive aspects of problem solving, other aspects like motivation and engagement are also important determinants of student success in problem solving. Therefore, it is likely that pedagogical approaches that motivate and engage students will also facilitate their transition from concepts to practical skills in programming courses, with the concomitant effect on student success.

6. Conclusions

In conclusion: 1) Average student performance in programming concepts is a good predictor of their average ability to apply that knowledge to write computer programs ($r^2 = 0.74$) indicating that adequate performance (70%) in programming concepts is *necessary* for students to write viable computer programs; 2) Some students (16%) cannot transfer conceptual knowledge and understanding into viable computer programs indicating that average adequate performance (70%) in programming concepts is *not sufficient* for students to write viable computer programs; 3) Adequate performance in *all* individual conceptual categories, and not just adequate *average* performance, is necessary to be able to write viable computer programs; 4) Given the correlations between performance in different conceptual categories, a subset of conceptual assessments consisting of repetition, classes, assignment operations and Java syntax is sufficient to predict students' ability to write viable programs ($r^2 = 0.78$); 5) Low values of r^2 may indicate that concepts taught and expected practical skills are not properly aligned. Thus regression analysis can be used to improve the alignment between concepts and skills facilitating student progress through the different cognitive levels in Bloom's taxonomy.

References

[1] Bloom, Benjamin S., Max B. Englehart, Edward J. Furst, Walter H. Hill, and David R. Krathwohl. (1956) *Taxonomy of Educational Objectives, the Classification of Educational Goals, Handbook I: Cognitive Domain*, edited by Benjamin S. Bloom. New York: McKay.

[2] Brooks, R.E. (1997). Towards a theory of the cognitive processes in computer programming. *International Journal of Man-Machine Studies* 9, 737–751. doi:10.1016/S0020-7373(77)80039-4

[3] Cabo, C. (2014). Transition from concepts to practical skills in computer programming courses: factor and cluster analysis. In *Proceedings of the 121st ASEE Annual Conference*. Washington, DC: ASEE.

[4] Cabo, C., and Lansiquot, R. D. (2014). Synergies between writing stories and writing programs in problem-solving courses. In *Proceedings of the 2014 IEEE Frontiers in Education Conference* (pp. 888-896). New York: IEEE.

[5] Cattell, R.B. (1966). The scree test for the number of factors. *Multivariate Behavioral Research* 1, 245-276.

[6] Davies, S.P. (1993). Models and theories of programming strategy. *International Journal of Man-Machine Studies* 39(2), 237–267. doi:10.1006/imms.1993.1061

[7] Deek, F. P., Kimmel, H., and McHugh, J. A. (1998). Pedagogical changes in the delivery of the first-course in computer science: Problem solving, then programming. *Journal of Engineering Education* 87(3), 313–320.

[8] Fincher, S. (1999). What are we doing when we teach programming? In *Proceedings of IEEE Frontiers in Education Conference,* (pp. 12A4/1-12A4/5). IEEE Press.

[9] Guttman, L. (1954). Some necessary conditions for common factor analysis. *Psychometrika* 19, 149-161.

[10] Horn, J.L. (1965). A rationale and test for the number of factors in factor analysis. *Psychometrika* 30, 179-185.

[11] Johnson-Laird, P. N. (1983). *Mental models*. Cambridge, UK: Cambridge University Press.

[12] Kay, J., Barg, M., Fekete, A., Greening, T., Hollands, O., Kingston, J., & Crawford, K. (2000). Problem-based learning for foundation computer science courses. *Computer Science Education* 10(2), 109–128. doi:10.1076/0899-3408(200008)10:2;1-C;FT109

[13] Krathwohl, David R. (2002). A Revision of Bloom's Taxonomy: An Overview. *Theory into Practice* 41, no. 4: 212-18.

[14] Lansiquot, R. D., and Cabo, C. (2010). The narrative of computing. In *Proceedings of World Conference on Educational Multimedia, Hypermedia and Telecommunications* (Toronto, Canada, June 28-July 01, 2010). ED-MEDIA 2010. AACE, Chesapeake, VA, 3655-3660.

[15] Lansiquot, R. D., and Cabo, C. (2011). Alice's Adventures in Programming Narratives. In C. Wankel and R. Hinrichs (Eds.), *Cutting-edge Technologies in Higher Education, Vol. 4: Transforming Virtual Learning*. Emerald, Bingley, UK, 311-331. DOI: http://dx.doi.org/10.1108/S2044-9968(2011)0000004016.

[16] Lorenzo-Seva, U. and Ferrando, P.J. (2006). FACTOR: A computer program to fit the exploratory factor analysis model. *Behavior Research Methods* 38, 88-91.

[17] Mayer, R.E. (1998). Cognitive, metacognitive, and motivational aspects of problem solving. *Instructional Science* 26, 49-63.

[18] Rist, R. S. (1995). Program structure and design. *Cognitive Science* 19, 507–562. doi:10.1207/s15516709cog1904_3

[19] Robins, A., Rountree, J., and Rountree, N. (2003). Learning and teaching programming. *Computer Science Education* 13, 2, 137-172.

[20] Spohrer, J. C., Soloway, E., and Pope, E. (1989). A goal/plan analysis of buggy Pascal programs. In Soloway, E., & Spohrer, J. C. (Eds.), *Studying the Novice Programmer* (pp. 355–399). Hillsdale, NJ: Lawrence Erlbaum. doi:10.1207/s15327051hci0102_4

[21] Soloway, E., and Spohrer, J. C. (Eds.). (1989). *Studying the novice programmer*. Hillsdale, NJ: Lawrence Erlbaum.

[22] Winslow, L. E. (1996). Programming pedagogy–A psychological overview. *SIGCSE Bulletin* 28(3), 17–22. doi:10.1145/234867.234872