Encouraging Generalization of Programming Concepts Outside a Student's "Home" Programming Language

David Whittinghill
Purdue University
dmwhittinghill@purdue.edu

David Nelson
Purdue University
davenelson@purdue.edu

Abstract

It is common for seasoned programmers and teachers to regard the concepts behind computer programming as generalizable as they are universal to all programming languages. The quintumvirate of variables, conditions, loops, functions, and arrays are in fact found almost universally in all modern programming languages, with object-oriented mechanisms seeing a similar proliferation. The assumption that these concepts are self-evident once taught and that programming students are therefore equipped to transfer this knowledge to other languages remains untested. This manuscript proposes an assessment intended to test this assumption. Thirty-four undergraduate students near the end of an introductory C++ programming course were assigned to groups and provided reference materials for languages not covered in the course. The students were presented with the course code of a short C++ program that generated random personal names and contained the five fundamental elements listed above. They were then asked to re-write the program on paper using a different programming language. The students were not graded on the accuracy of the translation but rather were surveyed as to their opinions of the exercise. Students rated the exercise as: 1) an exercise that depended upon what they learned in the class, 2) a good exercise in teamwork, 3) increasing their confidence that they could learn another programming language, and 4) an enjoyable experience.

Introduction

Computer programming education has become increasingly important in modern university curricula. Computer programming, a skill useful in and of itself, is only one aspect of this discipline's utility. The cognitive processes involved in computer programming – problem decomposition, logical process construction, quantitative construction of arguments [5] – provide benefit to students beyond a narrow technical competency and can be applied in a wide range of disciplines. For this reason, there has been increased focus by educators on not just computer programming but in algorithmic thinking or computational thinking; a mode of problem solving that involves approaches similar to those used to construct computer programs characterized by sequential, logical ordering of problem steps within a set of computational constraints [19].

Choosing the best approaches both for teaching programming, computational thinking, and assessing whether students are able to generalize the concepts they have learned has been an important area of research in programming education [13]. Researchers have examined the spectrum between first presenting the theoretical versus applied dimensions of programming [12]. Games [17], simulations [18], social constructivist systems [9], and even AI agents [15] have been developed to help facilitate improved learning and process outcomes. Motivational and psychological factors have also been studied [10] [8]. Though the literature describes a number of specific assessment tools that educators can implement [14] [2], more assessment tools are needed in order to provide them a well-rounded toolkit.

This paper describes a preliminary test of an assessment approach intended to give students practice applying what they have learned in an introductory C++ programming course to other languages beyond their "home" language the course is based upon. Students are given the source code of a working C++ program and asked to transcribe the program to an unfamiliar programming language within a 50-minute class session. Students must write their finished program on paper and can use a provided reference textbook as well as any other online materials they can find. The goals of the assessment were to create an activity that a) let the students apply the five cardinal programming constructs (variables, decisions, loop, arrays, and functions) in another programming language, b) provided an opportunity for teamwork, c) increased their confidence, and d) was enjoyable for the students. The authors describe the assessment technique and the results of a survey asking students how they felt about the assessment after completing it.

The post-assessment student survey results indicate that our assessment was: 1) appropriately dependent upon the course material, 2) a good opportunity to practice teamwork, 3) confidence building, 4) enjoyable. The open-ended qualitative responses showed that though impressions were positive overall, students would have preferred more time than the 50 minute class session allotted to them. The authors recommend future implementations of this assessment to be conducted in a computer laboratory so that students can attempt to compile the programs they create.

Review of the Literature

Psychological Factors in Programming Education

Computer science, and computer programming in particular, has been a major driver of technological progress in the modern era [1]. How best to teach students to program remains an unresolved question. I wealth of approaches exist that vary from the traditional lecture format, project-based learning, online courses, game-based learning, to many others [13]. Programming is a skill as well as an intellectual discipline and as such requires the student to not only study reference materials but to physically practice constructing programs as well. Moura et al. [11] examined active learning approaches in the teaching of introductory computer science courses. Her team defined a pedagogical distinction between deductive and inductive teaching approaches. In the deductive approach, the course is lead by theories and higher order concepts, from which the skills would then derive; in the inductive approach students began with hands-on examples and learned theories on a "need-to-know" basis, studying each theory as it became necessary. They discovered that students responded favorably to the active learning approach and that their motivation to learn was higher than in previous semesters that used the traditional teaching approach.

Learning motivation is a significant factor toward predicting student success in programming education, particularly with newcomers to the programming field [3]. Law has examined what motivating factors existed in an e-learning introductory programming course [8]. Among the more motivating attributes they discovered was that learning tasks that augment "individual attitude and expectation", and provide "clear direction", and "reward and recognition" were among the most motivating. Hawi [4] examined student beliefs as to why some students performed poorly in

an introductory programming course. Students attributed "lack of practice" and "learning strategy" among the important causal factors with "learning strategy" being attributed by all students who performed well in the course. McClure et al. [10] observed that students in any subject are likely to express a "self-serving bias" when suggesting the cause of their academic success or failure. The self-serving bias is the tendency for individuals to attribute their academic successes to internal factors (intelligence, hard work) and their failures to external factors (luck, unfair grading). This research clearly shows the relationship that exists between student success on an assessment and their belief that they have the ability to succeed. These findings demonstrate the need to test our assessment to elicit how students feel about our assessment approach and determine to what extent they believe it is engaging and fair.

Many novel approaches have been implemented to optimize and improve programming education. Lee [9] describe an approach for training teachers using a visual programming environment called Etoys in combination with pair programming and cognitive apprenticeship. Lee's approach was particularly interesting as it allowed teachers who may not have themselves been skilled in programming to become sufficiently proficient that they could create their own educational software applications thus enabling cross pollination of teaching ideas from outside the fields of computer science. An approach described by Villaroman et al. [18] involved using the Microsoft Xbox Kinect technology to provide students a completely different way of thinking about programming and problem solving. Rajabi constructed an artificially intelligent software agent that analyzed how software knowledge is acquired and attempted to derive optimal solutions. The presence of these innovative approaches indicate the widespread desire to discover groundbreaking new approaches to teaching computer programming

Computational Thinking

Computational thinking is the cognitive abstraction of programming that involves problem solving, designing systems, and understanding human behavior and cognition [19]. Cognitive thinking separates the process of constructing a step-by-step approach from the syntax of the programming language itself. Nathan [12] describes a continuum of teaching technique by the sequence in which *formalisms*, the concepts and theories of a discipline, are presented to the learner. He argues that a formalisms first (FF) approach is incorrect and, rather, a learner is best served by an inductive approach in which students are exposed to specific instances early in their education in order for concept generalization to derive from that direct experience. This mode of thinking necessarily advocates strongly for problem-based and inquiry-driven learning approaches. Scott [16] identifies today's students' apparent weakness in learning problem-solving skills and requirement abstraction (i.e., formalisms) and also advocates for a focus on late formalisms, early skills approach to teaching programming. [6] presents a design and test of a system called LEGCO which involves allowing students to construct conceptual meaning based on their interaction with geometric primitive objects in a C programming environment. Her research demonstrated that students performed better in this bottom-up environment than in the traditional Turbo C compiler environment. Kordaki et al. implemented a similar system called SORTING [7] that used a similar approach but focused instead on teaching sorting algorithms. Periera et al. [14], noting the difficulty students had in translating from a concept-first approach, implemented a problem-based learning approach that utilizes the concepts of cellular automata and fractals to provide students an opportunity to learn from an example-first approach. They found that their system did produce useful results providing students were first coached on the

fractals and cellular automata. These research results suggest strongly that exercises that focus on application over conceptualization are, perhaps paradoxically, more likely to result in students acquiring the ability to abstract and generalize their knowledge more quickly than if they are taught the concepts alone.

Methodology

Overview

Students were administered the assessment on the last day of the semester in lieu of a lecture. Concurrently, students had been working in self-selected groups together on a large coding project. At the beginning of class session, the students were instructed to assemble into the same groups as their coding project. All students were handed the source code for a text-based C++ program. The program utilized: variables, decisions, loops, arrays, and functions. A large stack of textbooks that covered a number of different programming languages was available at the front of the room. Each group was instructed that they had the entire course period to transcribe onto paper, using one of the unfamiliar languages, a version of the provided source code. The available programming languages were: Java, C#, Python, Objective-C, PL/SQL, and JavaScript. Students were allowed to talk freely, use the provided textbook, and utilize any online resources they liked using their personal laptops and smartphones.

Four proctors were available for the students as a resource to assist with technical questions and to monitor the teams' progress. The proctors were the course instructor, two graduate teaching assistants, and an industry professional from the games industry who is also an advisor to our academic department. Students were allotted 50 minutes to complete the task. They were instructed that the finished program did not need to compile, but that the results should capture the logic of the provided source code as closely as possible.

Upon completion of the activity, all subjects were emailed a hyperlink to an online survey that asked five questions. Students were expected to respond within three days in order to receive credit for participation. Student responses were stored in an online database (Qualtrics) and analyzed using the system's available statistical analysis tools.

Design

This study used a Posttest only design with four variables of interest: dependence on course pedagogy, teamwork, confidence, and enjoyment. All measurement data was collected from a posttest web survey that students filled out.

Survey

Four Likert-style questions were presented to the students with available responses of: Strongly Disagree, Disagree, Neither Agree Nor Disagree, Agree, and Strongly Agree. A fifth open-ended question was also included. The questions were as follows.

1. Having a strong grasp of programming fundamentals was necessary for this exercise.
2. I could not have done this exercise without the help of my team.
3. As a result of this exercise, I feel more confident about learning other new programming languages.

4. I enjoyed this exercise.
5. What did you like or dislike about this exercise?

Subjects
The entire subject pool consisted of 51 students, 34 of whom filled out the post-assessment survey. Subjects were rewarded with 10 points for participation. There was a total of 600 points awarded throughout the course of the semester.

Limitations
This study made no attempt to measure the accuracy of student work. The assessment was intended primarily as a participatory exercise that allowed students to practice generalizing the material they learned in the course into a different language context. Source code was written on paper rather than on a computer, which has a material effect on the viability of the source code the students produced (this is particularly true of Python).

Results

The observed results of the Likert questions are listed below.

1. Having a strong grasp of programming fundamentals was necessary for this exercise

#	Answer		Response	%
1	Strongly disagree		2	6%
2	Disagree		3	9%
3	Neither Agree nor Disagree		3	9%
4	Agree		14	41%
5	Strongly Agree		12	35%
	Total		34	100%

Statistic	Value
Min Value	1
Max Value	5
Mean	3.91
Variance	1.36
Standard Deviation	1.16
Total Responses	34

2. I could not have done this exercise without the help of my team

#	Answer		Response	%
1	Strongly disagree		1	3%
2	Disagree		6	18%
3	Neither Agree nor Disagree		4	12%
4	Agree		13	38%
5	Strongly Agree		10	29%
	Total		34	100%

Statistic	Value
Min Value	1
Max Value	5
Mean	3.74
Variance	1.35
Standard Deviation	1.16
Total Responses	34

3. As a result of this exercise, I feel more confident about learning other new programming languages

#	Answer		Response	%
1	Strongly disagree		1	3%
2	Disagree		3	9%
3	Neither Agree nor Disagree		8	24%
4	Agree		21	62%
5	Strongly Agree		1	3%
	Total		34	100%

Statistic	Value
Min Value	1
Max Value	5
Mean	3.53
Variance	0.68
Standard Deviation	0.83
Total Responses	34

4. I enjoyed this exercise

#	Answer		Response	%
1	Strongly disagree		0	0%
2	Disagree		4	12%
3	Neither Agree nor Disagree		12	35%
4	Agree		14	41%
5	Strongly Agree		4	12%
	Total		34	100%

Statistic	Value
Min Value	2
Max Value	5
Mean	3.53
Variance	0.74
Standard Deviation	0.86
Total Responses	34

The qualitative responses are not included in this manuscript as they were included in the study primarily to guide development of future iterations of the assessment. However, among the most common praise offered by students was that many students enjoyed the challenge the exercise

presented; while the most common critique was that they would have liked more time to finish their work.

The Likert data above show clear positive student reception along all four observed dimensions. All measured variables showed mean scores that were above the theoretical average.

Conclusion

The primary objective of this study was to determine student perceptions of our new assessment and whether they regarded the assessment as useful, fair, and enjoyable. Student reports show that the assessment succeeded along all four observed dimensions. This level of assessment is however only a first step. Now that the assessment has been determined by our students to be acceptable, further research is recommended to determine approaches for measuring the quality of the transcribed code produced by the students. Other important issues remain unaddressed. Is it best to have an assortment of different languages or perhaps only one different one? If different languages are used, how can assessment be fairly conducted given the different characteristics of different languages? Must teachers also be familiar with all the different languages used in the study in order to provide meaningful grades? Despite these unresolved questions, this assessment should be considered as, at the very least, a thought-provoking and enjoyable tool for programming educators to use for encouraging concept generalization.

Acknowledgements

The authors would like to thank Jeff Hanna of Volition Games for his assistance in proctoring the assessment session.

Bibliography

[1] Deitel, P., & Deitel, H. (2012). *C++ How to Program* (Vol. 8, p. 1104). Prentice Hall.

[2] Fürst, L., & Mahniĉ, V. (2012). A cooperative development system for an interactive introductory programming course. *World Transactions on Engineering and Technology Education*, *10*(2), 122–127.

[3] Guzdial, M., & Tew, A. E. (2006). Imagineering Inauthentic Legitimate Peripheral Participation : An Instructional Design Approach for Motivating Computing Education. In *Proceedings of the Second International Computing Education Research Workshop* (pp. 51–58). Canterbury, UK.

[4] Hawi, N. (2010). Causal attributions of success and failure made by undergraduate students in an introductory-level computer programming course. *Computers & Education*, *54*(4), 1127–1136.

[5] Henderson, P. B. (2003). Mathematical Reasoning in Software Engineering Education. *Communications of the ACM*, *46*(9), 45–50.

[6] Kordaki, M. (2010). A drawing and multi-representational computer environment for beginners' learning of programming using C: Design and pilot formative evaluation. *Computers & Education*, *54*(1), 69–87.

[7] Kordaki, M., Miatidis, M., & Kapsampelis, G. (2008). A computer environment for beginners' learning of sorting algorithms: Design and pilot evaluation. *Computers & Education*, *51*(2), 708–723.

[8] Law, K. M. Y., Lee, V. C. S., & Yu, Y. T. (2010). Learning motivation in e-learning facilitated computer programming courses. *Computers & Education*, *55*(1), 218–228.

[9] Lee, Y.-J. (2011). Empowering teachers to create educational software: A constructivist approach utilizing Etoys, pair programming and cognitive apprenticeship. *Computers & Education*, *56*(2), 527–538.

[10] McClure, J., Meyer, L. H., Garisch, J., Fischer, R., Weir, K. F., & Walkey, F. H. (2011). Students' attributions for their best and worst marks: Do they relate to achievement? *Contemporary Educational Psychology*, *36*(2), 71–81.

[11] Moura, I. C., & van Hattum-Janssen, N. (2011). Teaching a CS introductory course: An active approach. *Computers & Education*, *56*(2), 475–483.

[12] Nathan, M. J. (2012). Rethinking Formalisms in Formal Education. *Educational Psychologist*, *47*(2), 125–148.

[13] O'Grady, M. J. (2012). Practical Problem-Based Learning in Computing Education. *ACM Transactions on Computing Education*, *12*(3), 1–16.

[14] Pereira, H. B. D. B., Zebende, G. F., & Moret, M. a. (2010). Learning computer programming: Implementing a fractal in a Turing Machine. *Computers & Education*, *55*(2), 125–148.

[15] Rajabi, M., Aris, T. N. M., & Sulaiman, N. (2013). Computational problem solving architectural design based on multi - agent. *Journal of Theoretical and Applied Information Technology*, *58*(2), 311–318.

[16] Scott, A. S. (2010). *Using Flowcharts , Code and Animation for Improved Comprehension and Ability in Novice Programming Certificate of Research*. University of Glamorgan.

[17] Triantafyllakos, G., Palaigeorgiou, G., & Tsoukalas, I. a. (2011). Designing educational software with students through collaborative design games: The We!Design&Play framework. *Computers & Education*, *56*(1), 227–242.

[18] Villaroman, N., Rowe, D., Ph, D., & Swan, B. (2011). Teaching Natural User Interaction Using OpenNI and the Microsoft Kinect Sensor. In *Proceedings of the 2011 Conference on Information Technology Education* (pp. 227–231). West Point, NY: ACM Press.

[19] Wing, J. M. (2006). Computational Thinking. *Communications of the ACM*, *49*(3), 33–35.