Position Paper: Assessing Knowledge in Blocks-Based and Text-Based Programming Languages

Briana B. Morrison
School of Interactive Computing
Georgia Institute of Technology
Atlanta, GA USA
bmorrison@gatech.edu

Assessing student knowledge of programming concepts is a long studied, but many would say unsolved, problem. From the initial Empirical Studies of Programmers series *[1]* to McCracken's study *[2]* to Elliott-Tew's validated assessment of fundamental CS knowledge *[3]* many have attempted to precisely measure the knowledge and skills of students learning to program. All of the previous efforts have been centered on text based languages. Now we have another element to add to the mix – block-based programming languages. Can any of the existing assessment instruments work for both text and block based programming languages? Do we believe they will measure the same knowledge? Can existing assessments be modified and tailored for block-based languages? Are the two approaches equivalent, and would we even want to assess the same knowledge outcomes for both? These are but a few of the questions I will address in this position paper.

Keywords—assessment, outcomes, blocks, text

I. Background

Assessing programming knowledge of students is an often hotly debated topic among computing education researchers. In the original studies of programmer knowledge [1] both novices and expert programmers were studied to determine both their knowledge and skills and how novices could be encouraged to think and behave more like the expert programmers. However, no common assessment instrument was ever created to measure either knowledge or skills. In 2001, McCracken lead an ITiCSE working group to compare student knowledge and programming skills between institutions and across countries. The arguably simple task of programming a RPN calculator was attempted by over 200 students with much lower than expected results. The less than stellar results began a re-emphasis on assessment strategies – how were all these students able to pass a programming class and yet unable to complete the task?

Many science, technology, engineering and mathematics (STEM) disciplines have standard validated assessment tools that allow educators and researchers to accurately measure student learning and evaluate curricular innovations (e.g., [4], [5], [6]). However, computer science does not have a similar set of validated assessment tools, and practitioners and researchers must often devise their own instruments when they want to investigate student learning. The closest possible instrument may be the AP Computer Science A test, which is not readily available without cost and answers to all the free response questions are easily available on the internet.

In 2010, Elliott-Tew presented the FCS1, a language independent validated assessment instrument for measuring foundational CS1 knowledge in a language-independent manner. While much lauded at the time, the assessment has only been used one other time [7] and remains generally inaccessible today.

All of these assessments are based on text based programming languages. With the somewhat recent push toward block-based languages such as Scratch, AppInventor, and SNAP! for young novice programmers, there have been a few attempts at creating assessments of programming knowledge or computational thinking skills (e.g., [8], [9], [10]). In addition, an assessment for AP CS Principles, where instructors often uses block-based languages for the programming portion of the curriculum, is under development.

Yet, the assessment paths have never met. Is it possible to adapt the FCS1 for use with block-based languages? The test is implemented in pseudo-code which has been validated against student performance with procedural languages (Java, Python, MATLAB). Is it possible to adapt an existing blocks-based assessment for a text-based language?

II. Comparing Assessments

There are several issues to look at when examining the current assessments available for both text-based and block-based programming languages which include whether we should measure the same knowledge, the format of the test questions, and issues of scalability.

A. Knowledge Outcomes

There is a general consensus of the important concepts that should be covered in a CS1 course – sequence, conditionals, loops, data, and operators ([3], [9]. However, most existing assessments include these topics as well as others. Before creating an assessment which can be used with both block and text based languages, we must decide upon the knowledge outcomes to be tested. For example, [9] also includes testing knowledge of parallelism and events. Most educators would consider this inappropriate for a text-based language implementation of CS1. In most text-based language assessments, some form of procedure / parameter passing is

considered important, yet this does not often occur in block-based language assessments.

As researchers looking at the equivalency of both block and text based programming languages, we must decide what outcomes the assessments aim to measure and whether or not they should be identical or different for the separate environments. Perhaps there should be a common subset of topics for both, with separate concept areas added for each unique environment.

B. Format

While the question "What format should the assessments take?" is seemingly a simple one, it is anything but when it comes to block-based languages. In general most existing validated assessment instruments consist of multiple choice questions with perhaps some open-ended or free response questions. Currently none of the block-based language assessments use multiple choice questions. The development effort for the AP CS Principles assessment is currently attempting to convert its open-ended questions into multiple choice questions which can then be validated.

In text based programming assessments, common question forms include tracing (What is the output of this code segment?) or fill in the blank (What line(s) of code belong in the indicated space in order for the code to do X?). Neither of these forms of questions is easily adaptable for blocks-based languages. In [10], they used open-ended questions like "After this script executes which way will the sprite be facing?" and "What instructions must be added to the script to allow X to happen?" While these appear to be similar question types, both the creation and the grading process for them are very different. In a text-based language the questions and answers can be easily typed into a document. Not so with blocks-based languages – the image of the script and blocks must be captured and inserted into the document; this is usually a much more time consuming effort. In addition, each block-based language has a different set of blocks. Is there a general consensus of the subset of blocks which could be used for test questions? Is there an equivalent pseudo-blocks for these environments?

One possible question format which may work for both environments is a Parsons problem [11]. In a Parsons problem, lines of code (either text-based or blocks) are mixed-up and students are asked to put them into the correct order to solve the problem. Unfortunately, no validation of Parsons problems has yet occurred.

C. Scalability

The majority of all existing block-based language assessments involve having the student create new scripts to evaluate their knowledge. While this is completely acceptable for a teacher with a small or average sized-class, it is unfeasible for someone with a large class or researchers attempting to measure knowledge across thousands of students. While much headway has been made in the automatic grading of text-based program solutions, little is available for block-based program solutions. There is a pressing need for scalable, repeatable, validated assessments for block-based languages. Is it possible to create questions which measure similar knowledge and skills of the students without requiring them to create scripts / programs?

It is true that multiple choice questions have limitations and cannot accurately assess all aspects of a student's knowledge or their skills. We need the ability to assess a student's ability to design and correctly implement an algorithm and this is not readily testable using only multiple choice questions. Yet we cannot also score hundreds (or thousands) of independently constructed programs without resources similar to ETS (who pays for hundreds of program graders every summer to score the AP CS A free response questions). We need an intermediate solution which is easily automatically assessed and can still distinguish the knowledge level of the students.

III. Differences

There are proven educational learning differences between text based and blocks-based programming languages. In 2012, Margulieux et al. [12] replicated existing studies in other disciplines using subgoal labels with a block-based language (AppInventor). The participants were able to retain more information and perform the task more accurately when receiving subgoals than those who did not receive subgoal labels. However when the experiment was replicated with a text-based language [13] results were not as clear. There is an underlying question of why subgoals work as expected in a blocks-based environment, but not a text-based environment. Does this difference carry-over into assessments? Are there types of questions suitable for one and not the other? We must carefully examine all known learning differences between the environments while designing assessments.

IV. Future Research Questions

When it comes to assessment of student programming knowledge in text based and block based languages, there are more questions than answers at this point in time. While text-based assessments exist, none are ideal and none have been adapted or "translated" to block-based languages. None of the existing block-based assessments have been validated or are scalable. Most involve the creation of new scripts to assess student knowledge which is generally labor intensive to grade with many different possible correct solutions. When looking at block programming environments, an important topic to be considered is assessment of student knowledge. If we hope for educators to adopt and use block-based environments for teaching CS1 we must also provide them with assessment instruments to measure student knowledge and skills.

References

[1] E. Soloway and K. Ehrlich, "Empirical studies of programming knowledge," *Softw. Eng. IEEE Trans.*, no. 5, pp. 595–609, 1984.

[2] M. McCracken, V. Almstrum, D. Diaz, M. Guzdial, D. Hagan, Y. B.-D. Kolikant, C. Laxer, L. Thomas, I. Utting, and T. Wilusz, "A multi-national, multi-institutional study of assessment of programming skills of first-year CS students," in *Working group reports from ITiCSE on Innovation and technology in computer science education*, Canterbury, UK, 2001, pp. 125–180.

[3] A. E. Tew, "Assessing fundamental introductory computing concept knowledge in a language independent manner," 2010.

[4] D. Hestenes, M. Wells, G. Swackhamer, and others, "Force concept inventory," *Phys. Teach.*, vol. 30, no. 3, pp. 141–158, 1992.
[5] C. D'Avanzo, "Biology concept inventories: overview, status, and next steps," *BioScience*, vol. 58, no. 11, pp. 1079–1085, 2008.
[6] J. C. Libarkin and S. W. Anderson, "Assessment of learning in entry-level geoscience courses: Results from the Geoscience Concept Inventory," *J. Geosci. Educ.*, vol. 53, no. 4, p. 394, 2005.
[7] I. Utting, A. E. Tew, M. McCracken, L. Thomas, D. Bouvier, R. Frye, J. Paterson, M. Caspersen, Y. B.-D. Kolikant, J. Sorva, and T. Wilusz, "A Fresh Look at Novice Programmers' Performance and Their Teachers' Expectations," in *Proceedings of the ITiCSE Working Group Reports Conference on Innovation and Technology in Computer Science Education-working Group Reports*, New York, NY, USA, 2013, pp. 15–32.
[8] S. Grover, S. Cooper, and R. Pea, "Assessing Computational Learning in K-12," in *Proceedings of the 2014 Conference on Innovation & Technology in Computer Science Education*, New York, NY, USA, 2014, pp. 57–62.
[9] K. Brennan and M. Resnick, "New frameworks for studying and assessing the development of computational thinking," in *Proceedings of the 2012 annual meeting of the American Educational Research Association, Vancouver, Canada*, 2012.
[10] O. Meerbaum-Salant, M. Armoni, and M. (Moti) Ben-Ari, "Learning computer science concepts with Scratch," *Comput. Sci. Educ.*, vol. 23, no. 3, pp. 239–264, Sep. 2013.
[11] D. Parsons and P. Haden, "Parson's Programming Puzzles: A Fun and Effective Learning Tool for First Programming Courses," in *Proceedings of the 8th Australasian Conference on Computing Education - Volume 52*, Darlinghurst, Australia, Australia, 2006, pp. 157–163.
[12] L. E. Margulieux, M. Guzdial, and R. Catrambone, "Subgoal-labeled instructional material improves performance and transfer in learning to develop mobile applications," in *Proceedings of the ninth annual international conference on International computing education research*, 2012, pp. 71–78.
[13] Morrison, Briana B., Margulieux, Lauren E., and Guzdial, Mark, "Subgoals, Context, and Worked Examples in Learning Computing Problem Solving," in *ICER 2015*, 2015.