Developing a Validated Assessment of Fundamental CS1 Concepts

Allison Elliott Tew and Mark Guzdial
School of Interactive Computing
Georgia Institute of Technology
Atlanta, GA 30332-0760
{allison, guzdial}@cc.gatech.edu

ABSTRACT

Previous studies of student programming ability have raised questions about students' ability to problem solve, read and analyze code, and understand introductory computing concepts. However, it is unclear whether these results are the product of failures of student comprehension or our inability to accurately measure their performance. We propose a method for creating a language independent CS1 assessment instrument and present the results of our analysis used to define the common conceptual content that will serve as the framework for the exam. We conclude with a discussion of future work and our progress towards developing the assessment.

Categories and Subject Descriptors

K.3.2 [**Computers and Education**]: Computer and Information Science Education—*computer science education, curriculum*

General Terms

Experimentation, measurement

Keywords

Assessment, CS1, programming, validity

1. INTRODUCTION

Measuring student learning is fundamental to any educational endeavor. Many STEM disciplines have standard validated assessment tools that allow educators and researchers to use the instruments to accurately measure student learning and evaluate curricular innovations (e.g., [10, 4, 14]). However, computer science does not have a similar set of validated assessment tools, and practitioners must devise their own instruments each time they want to investigate student learning.

Consider the following recent studies of student programming ability with carefully designed, albeit non-validated, assessment plans:

A well-cited study by McCracken, et al asked students to write a program, in a laboratory setting, to build a simple calculator [19]. Students performed much worse than expected earning an average of only 20.8% of the possible points on the assessment. The researchers concluded that students did not possess the basic programming skills expected at the end of the introductory sequence. They reasoned that the students lacked problem solving ability and had difficulty abstracting a potential solution from the problem description.

Lister, et al explored an alternative hypothesis for the students' poor performance in the McCracken study by assessing students' code comprehension and tracing ability, claiming these were prerequisite skills to problem solving [16]. The assessment consisted of twelve multiple choice questions (MCQs) focusing on arrays and iteration. Overall, an average of 60% of the students answered the questions correctly, and the researchers concluded that students struggled with the preliminary, basic skills of reading and analyzing code.

Tew, et al investigated the impact of different CS1 courses on the conceptual knowledge students demonstrate at the end of the course [24]. Their multiple-choice question assessment found statistically significant differences in knowledge understanding after CS1 in several fundamental introductory concepts (e.g., conditional, array, sorting), with only 42.3% of the students answering questions about the introductory material correctly. They repeated their assessment at the end of a common second course and found that the differences in concept understanding were no longer distinguishable.

A common theme among these studies is that students are not performing as well as we would expect and are not demonstrating mastery of fundamental material, often considered to be some of the most basic ideas covered in the introductory curricula. Rather than providing a clear consensus or direction, these studies raise a number of questions. Do students comprehend the computing concepts we cover in our introductory courses? Are students able to demonstrate programming problem solving ability? Is code comprehension a prerequisite skill for other programming activities? Perhaps the study results are more indicative of our lack of precise measures, rather than an accurate measure of students' ability and knowledge.

Valid measures would enable us to know for sure that we are measuring the results of student learning, would enable direct comparison of pedagogical approaches, and would permit investigation of curricular innovations. Computing education and research suffer from the lack of such instruments, and we seek to develop a validated assessment for CS1 that would be widely applicable across curricular approaches.

This paper begins with a argument for the importance of valid assessment instruments in computing and then proposes a method for developing such an instrument in Section 3. Our current progress towards developing the assessment is presented in the following sections. Section 4 contains the results of our document analysis used to identify common introductory concepts, and the test specification and question development is outlined in Section 5. We conclude with a discussion of future work.

2. VALIDITY OF ASSESSMENT

Validity is a measure of "how well the test serves the purpose for which it is used" [15, p. 621]. In other words, validity is the evidence that assures us that questions about a particular concept are indeed measuring that concept. For instance, a question about arrays should require a student to have knowledge about arrays, but should not require knowledge about another concept, such as recursion. In addition, it is important that the question cannot be answered correctly without knowledge of arrays.

In general, there are two classes of evidence used to support validity claims. *Content* related evidence ensures that the assessment's content appropriately operationalizes the constructs it is intended to measure. A test designed to assess CS1 knowledge would therefore identify a number of topics to be measured, and its content validity would be determined by whether the set of topics is a reasonable operationalization of CS1.

Construct related evidence provides the second set of support for validity. A construct is "the concept or the characteristic that a test is designed to measure" [2, p. 173]. Empirical analysis of responses to individual questions provides evidence that the items in the assessment are indeed measuring the desired constructs, rather than something more or less than intended. Together, content and construct validity enable a test developer to provide evidence that the instrument is accurately measuring student knowledge and skills as intended.

Current assessment practices in computing are limited in a number of ways. Most of the measures used by researchers in our field, like the questions used in the McCracken, Lister and Tew studies mentioned previously, are not validated, so we cannot be sure that we are measuring the desired outcomes. On the other hand, the CS Advanced Placement exam has been validated but is tied to a particular programming language and is therefore not widely applicable across different approaches or courses. Similarly, Decker [5] developed an assessment for the introductory sequence of programming courses, also tied specifically to the Java programming language. The instrument was developed and tested at a single institution and therefore its validity claims cannot be generalized beyond that context. Other validated tests in our field are end of program assessments (e.g., GRE Subject Test, Major Field Test) and have psychometric or predictive aims that do not focus on assessing learning.

Building upon the success of concept inventories in physics and other STEM fields, Zilles, et al have begun development on a set of inventories for computing [8]. They assembled a group of experts and are using the Delphi process to identify sets of difficult and important concepts to develop assessments in three areas: discrete math, logic design, and programming. However unlike natural sciences, computer science knowledge has no real world analog, and it remains an open research question as to whether a set of common misconceptions, upon which concept inventories are built, can be identified for computing.

Our work differs from these efforts in that we aim to create a rigorously validated exam that could be widely adopted and used in any introductory CS course. Our goal is to create an exam to measure understanding of fundamental computing concepts, independent of programming language, that would not be overly biased by any particular pedagogical paradigm.

3. METHOD FOR DEVELOPING A VALIDATED CS1 ASSESSMENT

To build our assessment instrument we are adapting the standard educational test development guidelines [2]. In order to achieve language independence, we have augmented these procedures with an additional step. The steps in the resulting process are outlined below:

1. Define Conceptual Content
2. Expert Review of Test Specification
3. Build Test Bank of Questions
4. Verify Language Independence
5. Pilot Questions
6. Establish Validity
7. Establish Reliability

The first step in test development is to establish the purpose and definition of the test—what is to be measured and what the scores mean. The test specification includes the definition of the conceptual content, or constructs, that are to be measured, the format of the questions, and the scoring procedures. The test specification should be reviewed by a panel of experts to provide content validity evidence and ensure that all constructs are adequately represented and extraneous constructs are not included.

After the test specification has been developed and verified, a test bank of questions should be developed to cover all constructs identified in the specification. To achieve language independence for a CS1 exam, we utilize a verbose psuedocode as the exam programming language. Piloting of the questions in both pseudocode and the language of instruction, for example Java or Python, then takes place. This testing is required to ensure students are able to appropriately transfer understanding from their programming language of instruction to pseudocode. Pilot tests also examine the suitability of the questions, allowing necessary revisions to be made prior to the selection of the final candidate questions.

The last stages of test development are empirical studies of individual responses to establish validity and reliability. Validity testing ensures that the test is measuring the intended constructs, and reliability testing verifies that the results are consistent over repeated tests, and thus are dependable.

The following sections of the paper present our results to date. We have completed work defining the conceptual content of the assessment and have built the test bank of questions.

4. DEFINING THE CONTENT

Given our goal of developing a widely applicable CS1 assessment, our strategy for defining content was to identify concepts that a wide variety of introductory courses and approaches had in common. We chose textbooks as the external artifact representing the content of a course because other measures such as course syllabi or assignments are not feasible to analyze on a large scale. We began by conducting a document analysis of the table of contents of the two most widely adopted CS1 textbooks from each of the major publishers of computing textbooks (Addison Wesley, Thomson/Course Technology, Franklin Beedle & Associates, McGraw Hill, Prentice Hall, Wiley & Sons)—12 books in total [3, 6, 11, 12, 13, 17, 18, 20, 22, 23, 25, 26].

Topics listed in the table of contents were aggregated into a list, noting which concepts were covered by which texts. The goal of this bottom-up approach was to identify the set of topics most commonly covered in CS1 courses, as specified by textbooks faculty chose to adopt. However with the increasing breadth in introductory textbooks, the topic list quickly became unwieldy with over 400 concepts, ranging from low level concepts such as byte code and computer architecture to advanced topics traditionally covered later in the curriculum (e.g., relational databases and multi-threaded processes).

We used the framework of the Computer Science volume of Computing Curricula 2001 [1] to revise our initial list of topics. CC2001 provides guidelines for the conceptual content to be covered in the introductory year sequence of computing courses. By providing a variety of models and pedagogical approaches to achieve these goals, the guidelines do not designate any concepts specific to the first or second CS course. Although a list of CS1 topics is not specified, the framework did enable revisions by providing a high level organization for the concepts identified in the first phase of analysis. We chose to eliminate any concepts outside of the scope of the introductory sequence. We further narrowed the intended scope of our assessment by concentrating on the identified concepts that were in the programming fundamentals (PF1, PF3, and PF4) and object-oriented programming (PL6) areas while removing categories such as discrete structures, algorithms and complexity, and software engineering.

Unfortunately, the resulting list of 188 concepts was still too large to be practical for test development. We further refined the list by analyzing the content of canonical texts representing each of the common introductory approaches (objects-first [13, 6], functional-first [7], and imperative-first [26]). A concept was included in this step of revision if it was covered by all of the texts or excluded by only one of the canonical texts. The list of fundamental computing concepts common across languages and pedagogical approaches is listed in Table 1.

The topics in Table 1 have been refined to a scope that fits within the material traditionally covered in CS1. However it is impractical to sufficiently evaluate student knowledge of each of these 29 concepts in a single test setting. We therefore performed additional analysis and synthesis with the aim of generating a small handful of constructs that were

Table 1: Common Fundamental CS1 Concepts

Concept	Lewis & Loftus	Deitel & Deitel	Felleisen et al	Zelle
Variable	x	x	x	x
Simple I/O	x	x		x
Recursion	x	x	x	x
EXPRESSIONS				
Mathematical Operators	x	x	x	x
Relational Operators	x	x	x	x
Logical Operators	x	x	x	x
Assignment	x	x	x	x
CONTROL STRUCTURES				
Selection Statement (if/else)	x	x		x
Definite Loop (for)	x	x	x	x
Indefinite Loop (while)	x	x		x
Nested Loops	x	x		x
FUNCTIONS/METHODS				
Definition	x	x	x	x
Parameters - Pass by Value	x	x	x	x
Return Values	x	x	x	x
DATA TYPES & STRUCTURES				
Primitive Data Types	x	x	x	x
Integer	x	x	x	x
Floating Point	x	x		x
Boolean	x	x	x	x
String	x	x	x	x
Array	x	x	x	x
Tree	x	x	x	
OBJECT-ORIENTED PROGRAMMING				
Object/Class	x	x	x	x
Constructor	x	x	x	x
Instance/Local Variables	x	x	x	x
Accessor Methods	x	x	x	
Mutators Methods	x	x	x	
Encapsulation	x	x	x	x
Inheritance	x	x		x
Polymorphism	x	x		x

amenable to testing. A number of basic concepts were combined into one *fundamentals* construct that includes all of the basic semantic topics (e.g., variables, assignment, mathematical expressions). The primitive data type concepts (e.g., integer, boolean) provide useful information for the kinds of data commonly available for manipulation in test questions, but we chose not to dedicate separate questions to these topics. Procedures for processing simple input and output are often very language specific, so this topic was removed over concern for generalizability across languages and paradigms. Finally, in order to avoid biasing a particular paradigm and to limit the scope of constructs to those most fundamental and widely applicable across any introductory approach, we chose to limit the object construct to the basics of class definitions and method calls. The final list of constructs which serve as the basis of the test specification are as follows:

- Fundamentals (variables, assignment, etc.)
- Logical Operators
- Selection Statement (if/else)
- Definite Loops (for)
- Indefinite Loops (while)
- Arrays
- Function/method parameters
- Function/method return values
- Recursion
- Object-oriented Basics (class definition, method calls)

5. TEST SPECIFICATION AND QUESTION DEVELOPMENT

A test specification is a detailed description of the instrument that specifies the percentage of questions dedicated to each construct, the question format, and the scoring procedures [2]. We have chosen to weight each construct equally with 10% of the questions devoted to each topic and have elected to use a multiple-choice question format. MCQs, when constructed carefully, can provide the same information about conceptual knowledge as short answer or open response questions with significant advantages in test administration and scoring [9]. We have also specified that test scores should be criterion-referenced, interpreted based on individual performance and not rated relative to the performance of peers.

A group of experts in CS education was empaneled to review our test specification. Specifically they provided feedback on the list of constructs to be tested, the standardized multiple-choice question format, and the scoring method to be used. An initial draft of sample questions was provided to help concretize the testing constructs. Based on their feedback, the operational definitions for the constructs were finalized and question development began.

In order to evaluate different kinds of conceptual understanding, three different types of questions about each construct were developed. Definitional questions explore the student's general understanding of a construct. Tracing questions examine a student's ability to predict execution of code using a particular concept (e.g. the value of a variable after loop completes execution). Code completion questions are fill-in-the blank type questions to evaluate a student's ability to write code. For each construct, we built multiple versions of each type of question for the test bank[1].

6. FUTURE WORK AND CONCLUSION

At this stage we have specified the content of our CS1 assessment and the nature of the test's format. We have also developed the bank of questions which cover the common CS1 concepts identified through the analysis of introductory texts. Our next steps in the process are to conduct pilot studies which will allow us to gather evidence for language independence and construct validity. We briefly outline the plans for these studies below.

To achieve the goal of having a language independent CS1 assessment that can be widely adopted, we have an additional burden of investigating the effect of pseudocode as the examination language. We will conduct this inquiry in two stages. The first will compare students' answers to open-ended versions of the questions written in the programming language of their CS1 instruction. Student answer patterns between classes of languages and paradigms will then be compared for similarities and differences in their responses. Common student answers, particularly the incorrect ones, suggest common errors regardless of programming language, and they will be used to generate and verify distractors (i.e., wrong answers) for the multiple-choice questions on the assessment.

The second stage in piloting our assessment will be comparing student performance on analogous versions of the assessment, one written in pseudocode and one written in their CS1 programming language. Ideally, with a brief introduction to the pseudocode, students should be able to transfer their knowledge of computing concepts to the syntax of the pseudocode and perform comparably on the two versions of the assessment.

Empirical analysis of item responses gathered during the pilot phases may suggest revisions. Final candidate questions for the instrument will be selected after any necessary revisions are made. This candidate instrument will then be deployed at multiple institutions for full-scale validity and reliability testing. Construct validity evidence will be gathered by correlating assessment scores with other measures of CS1 learning, such as final exam scores. Additional data will be collected through think aloud protocols to ensure students interpret questions as intended. Reliability of the instrument will be measured using split-half testing. The instrument will be divided into two equivalent tests, administered at 1–2 week intervals, and correlations between the two scores computed.

The method presented above outlines essential steps towards achieving the goal of producing a validated and reliable assessment instrument for introductory computing concepts. The creation of such assessment tools, similar to those available in other STEM disciplines, is an important endeavor for our developing field. As Moss, et al state "assessment practices do far more than provide information, they

[1]Unfortunately, until we have established the validity of the assessment instrument, exam questions must remain private and cannot be published. This prevents potentially biasing participants involved in the validation studies.

also shape people's understanding about what is important to learn, what learning is, and who learners are" [21, p. 111]. No one assessment could, or should, sufficiently define the scope of computing education, so partnerships with practitioners and researchers encouraging the development of new validated assessment instruments are vital for our discipline.

7. ACKNOWLEDGMENTS

This material is based upon work supported by the National Science Foundation under Grant Nos. 0512213 and 0634629. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

8. REFERENCES

[1] Computing curricula 2001. *Journal on Educational Resources in Computing*, 1(3es):1–240, 2001.

[2] American Educational Research Association, American Psychological Association, and National Council on Measurement in Education. *Standards for educational and psychological testing*. American Educational Research Association, Washington, DC, 1999.

[3] J. Cohoon and J. Davidson. *Java 5.0 Program Design*. McGraw Hill, Boston, MA, 2006.

[4] C. H. Crouch and E. Mazur. Peer instruction: Ten years of experience and results. *American Journal of Physics*, 69(9):970–977, September 2001.

[5] A. M. Decker. *How Students Measure Up: An Assessment Instrument for Introductory Computer Science*. PhD thesis, University at Buffalo (SUNY), Buffalo, NY, 2007.

[6] H. Deitel and P. Deitel. *C++: How to Program*. Prentice Hall, Upper Saddle River, NJ, 5th edition, 2005.

[7] M. Felleisen, R. B. Findler, M. Flatt, and S. Krishnamurthi. *How to Design Programs: An Introduction to Programming and Computing*. MIT Press, Cambridge, MA, 2001.

[8] K. Goldman, P. Gross, C. Heeren, G. Herman, L. Kaczmarczyk, M. C. Loui, and C. Zilles. Identifying important and difficult concepts in introductory computing courses using a Delphi process. In *SIGCSE '08: Proceedings of the 39th SIGCSE Technical Symposium on Computer Science Education*, pages 256–260, 2008.

[9] T. M. Haladyna. *Developing and validating multiple-choice test items*. Lawrence Erlbaum Associates, Inc., Mahwah, NJ, 3rd edition, 2004.

[10] D. Hestenes, M. Wells, and G. Swackhamer. Force concept inventory. *The Physics Teacher*, 30:141–158, March 1992.

[11] C. Horstmann. *Java Concepts*. John Wiley and Sons, Hoboken, NJ, 4th edition, 2005.

[12] C. Horstmann. *Big Java*. John Wiley and Sons, Hoboken, NJ, 2nd edition, 2006.

[13] J. Lewis and W. Loftus. *Java Software Solutions (Java 5.0 version): Foundations of Program Design*. Addison Wesley, Boston, MA, 4th edition, 2005.

[14] J. C. Libarkin and S. Anderson. Assessment of learning in entry-level geoscience courses: Results from the geoscience concept inventory. *Journal of Geoscience Education*, 53:394–401, 2005.

[15] E. F. Lindquist, editor. *Educational measurement*. American Council on Education, Washington, D.C., 1951.

[16] R. Lister, E. S. Adams, S. Fitzgerald, W. Fone, J. Hamer, M. Lindholm, R. McCartney, J. E. Moström, K. Sanders, O. Seppälä, B. Simon, and L. Thomas. A multi-national 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, 2004.

[17] D. S. Malik. *C++ Programming: From Problem Analysis to Program Design*. Thompson Course Technology, Boston, MA, 2nd edition, 2004.

[18] D. S. Malik. *Java Programming: From Problem Analysis to Program Design*. Thompson Course Technology, Boston, MA, 2nd edition, 2006.

[19] 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. *SIGCSE Bulletin*, 33(4):125–180, 2001.

[20] R. Mercer. *Computing Fundamentals with Java*. Franklin Beedle and Associates, Wilsonville, OR, 2002.

[21] P. A. Moss, B. J. Girard, and L. C. Haniford. Validity in Educational Assessment. *Review of Research in Education*, 30(1):109–162, 2006.

[22] W. Savitch. *Java: An Introduction to Problem Solving and Programming*. Prentice Hall, Upper Saddle River, NJ, 4th edition edition, 2005.

[23] W. Savitch. *Problem Solving with C++: The Object of Programming*. Addison Wesley, Boston, MA, 5th edition edition, 2005.

[24] A. E. Tew, W. M. McCracken, and M. Guzdial. Impact of alternative introductory courses on programming concept understanding. In *ICER '05: Proceedings of the 2005 International Workshop on Computing Education Research*, pages 25–35, 2005.

[25] C. T. Wu. *Intro to Object Oriented Programming using Java*. McGraw Hill, Boston, MA, 4th edition, 2006.

[26] J. M. Zelle. *Python Programming: An Introduction to Computer Science*. Franklin Beedle, Wilsonville, OR, 2004.