ASSESSING FUNDAMENTAL INTRODUCTORY COMPUTING CONCEPT KNOWLEDGE IN A LANGUAGE INDEPENDENT MANNER

A Dissertation
Presented to
The Academic Faculty

by

Allison Elliott Tew

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in
Computer Science

School of Interactive Computing
Georgia Institute of Technology
December 2010

ProQuest LLC
789 East Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106-1346

ASSESSING FUNDAMENTAL INTRODUCTORY COMPUTING CONCEPT KNOWLEDGE IN A LANGUAGE INDEPENDENT MANNER

Approved by:

Mark Guzdial, Committee Chair
School of Interactive Computing
Georgia Institute of Technology

Amy Bruckman
School of Interactive Computing
Georgia Institute of Technology

Stephen Cooper
Department of Computer Science
Stanford University

Sally A. Fincher
School of Computing
University of Kent

W. Michael McCracken
School of Computer Science
Georgia Institute of Technology

Date Approved: August 20, 2010

To my parents,

who have always encouraged me to follow my dreams.

And to John,

who has dared to dream along with me.

ACKNOWLEDGEMENTS

Two roads diverged in a yellow wood,
And sorry I could not travel both
And be one traveler, long I stood
...
I shall be telling this with a sigh
Somewhere ages and ages hence:
Two roads diverged in a wood, and I—
I took the one less traveled by,
And that has made all the difference.

Robert Frost

Most people will readily admit that I have taken the scenic, less traveled, path to complete my degree. However, this has enabled me to be surrounded by an incredible group of friends, family, colleagues and mentors along the way.

I'd like to begin by thanking my advisor, Mark Guzdial, who has been my mentor and colleague for over 15 years. He has led me to great successes and stuck with me through my many failures. All along the way he has inspired me with his passion, enthusiasm, and undying optimism. Mike McCracken started me on this journey many, many years ago by introducing me to research as an undergraduate student and prodding me to consider applying to PhD programs—the first time around. Amy Bruckman was a strong ally for the start of a Computer Science Education research group at Georgia Tech. She always believed that the field was important and supported my work even as it diverged from her interests and expertise.

My external committee members, Sally Fincher and Steve Cooper, are two mentors I discovered while participating in the Scaffolding Research in Computer Science Education project. Sally gave me a crash course in how to do high quality computing education research, but more importantly turned the light back on. My research

career today is thanks in no small part to Josh Tenenberg and to her. They showed me the way and inspired me to want to participate in and contribute to the research endeavor again. Sally and Josh continue to provide outstanding counsel, and I am forever indebted and grateful. Steve Cooper was one of the earliest proponents of my thesis project. His support, advice, and proselytizing have continued to help shape my work through today.

This research would not have been possible without the generous cooperation of the many faculty and students who have allowed me into their classrooms. I would like to thank the faculty who taught the introductory courses at Georgia Tech (Kristin Marsicano, Cedric Stallworth, Jay Summet, and David Smith), the University of Georgia (Julia Couto and Chris Plaue), and the University of British Columbia (Kurt Eiselt), as well as AP teachers in Atlanta area high schools. In addition, I would like to thank my collaborators on this project, John Kim, Dannon Baker, David Joyner, and Bobby Matthews. This work was made possible and funded in part by the National Science Foundation (CISE #0306050, CCLI-ASA #0512213, CPATH CB #0829601, BPC-A #0634629).

A Ph.D. is an oddly solitary path, yet one that cannot possibly be traversed alone. I am indebted to friends and colleagues both inside and outside of Georgia Tech who have walked beside me along the way. While it is impossible to mention the contributions of everyone here, I count myself blessed to have been surrounded by such a wonderful and diverse group of individuals. Thanks to Leigh Waguespack, Jimmy Yang, Jason Wright, James Shamiyeh, Jay Dolce, and Sam Panchal for being there and understanding when I couldn't be. Colleen Kehoe, Annie Anton, and Brad Topol, my colleagues and lab-mates from before, have waited patiently too many years to count for me to finish. Thanks for your support, advice, and encouragement – back when we shared cube walls and again now as I entered the home stretch. To my colleagues from my "former life," thanks for letting me leave to pursue this degree

and for sticking with me through all these years. I hope I have made you proud. To my fellow students in the Contextualized Support for Learning Lab and the Learning Sciences and Technology Group, past and present, thanks for welcoming me into the fold, teaching me the ropes, and helping me focus my ideas and clarify my thoughts. Brian Landry, Brian O'Neill, Sarita Yardi, Jill Dimond, and Tammy Clegg, you have each brought me joy, laughter, and happiness, and for that I thank you.

Tracy Westeyn and Briana Morrison, two dear friends that I made along the way, thank you for all of your support, encouragement, and advice. I hope I was able to give as much as I received. To my "Allison", Kurt Eiselt — I have been guided by your hand and a part of your family for so long, I don't know any other way. Thank you for being the voice of reason, the older brother I've always wanted, and a truly great friend. To my intellectual soul mate, Brian Dorn, thanks for everything. My work has been made infinitely better, I have remained sane, and I have even enjoyed my time as a grad student because of you. I would not have survived the program without each of you; your friendship has been an invaluable resource in my journey.

Finally, I would like to thank my family and friends for your support and encouragement throughout the process. I appreciate your patience and understanding as I continued to pursue this dream, and words cannot begin to express how grateful I am that you were with me for this final milestone. I want to thank my father for giving me such big shoes to fill but always holding my hand to show me the way. My mother has always patiently encouraged me to pursue my own dreams, even when they were slow to come to fruition or even harder to understand.

And to John — thanks for standing by my side along the way, always believing that it was possible, and daring to jump off the cliff with me now. May we always chose new adventures and encourage each other to dream big.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

Measuring student learning is fundamental to any educational endeavor. A primary goal of many computer science education projects is to determine the extent to which a given instructional intervention has had an impact on student learning. However, the field of computing lacks valid and reliable assessment instruments for pedagogical or research purposes. Without such valid assessments, it is difficult to accurately measure student learning or establish a relationship between the instructional setting and learning outcomes. The goal of assessment research in computer science is to have valid ways of measuring student conceptions of fundamental topics, which will enable both research into how understanding of knowledge in the domain develops as well as enable curricular innovation and reform grounded in this knowledge.

My dissertation work focused on three questions regarding assessment of introductory concepts in computer science. How can existing test development methods be applied and adapted to create a valid assessment instrument for CS1 conceptual knowledge? To what extent can pseudo-code be used as the mechanism for achieving programming language independence in an assessment instrument? And to what extent does the language independent instrument provide a valid measure of CS1 conceptual knowledge?

I developed the Foundational CS1 (FCS1) Assessment instrument, the first assessment instrument for introductory computer science concepts that is applicable across a variety of current pedagogies and programming languages. I applied methods from educational and psychological test development, adapting them as necessary to fit the disciplinary context. I conducted think aloud interviews and a large scale empirical

study to demonstrate that pseudo-code was an appropriate mechanism for achieving programming language independence. Student participants were able to read and reason in the pseudo-code syntax without difficulty and were able to transfer conceptual knowledge from their CS1 programming language to pseudo-code. Finally, I established the validity of the assessment using a multi-faceted argument, combining interview data, statistical analysis of results on the assessment, and exam scores.

The contributions of this research are: (1) An example of how to bootstrap the process for developing the first assessment instrument for a disciplinary specific design-based field. (2) Identification that although it may not be possible to correlate scores between computer science exams created with different measurement goals, the validity claims of the individual assessments are not diminished. (3) A demonstration that novice computing students, at an appropriate level of development, can transfer their understanding of fundamental concepts to pseudo-code notation. (4) A valid assessment of introductory computing concepts for procedurally-based introductory computing courses taught in Java, Matlab, or Python at the university level.

CHAPTER I

INTRODUCTION AND MOTIVATION

Measuring student learning is fundamental to any educational endeavor. 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., (Hestenes, Wells, & Swackhamer, 1992; Crouch & Mazur, 2001; Libarkin & Anderson, 2005)). 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.

Consider the following recent studies of student programming ability with carefully designed 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 (McCracken et al., 2001). 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. They assessed students' code comprehension and tracing ability, claiming these were prerequisite skills to problem solving (Lister et al., 2004). 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

Table 1: Significant Differences in Pre-Test Concepts – Spring 2005

Concept	% Correct		α
	'Computing'	'Engineering'	
Iteration	25%	44%	0.025
Conditional	18%	46%	0.001
Binary Search Tree	84%	69%	0.05
Array	58%	30%	0.001
Sorting	36%	21%	0.025

correctly, and the researchers concluded that students struggled with the preliminary, basic skills of reading and analyzing code.

In previous work, I investigated the impact of alternative approaches to introductory computing by considering the questions of what students bring to their second class in computing and how the outcomes differ depending on the students alternative first course (Tew, McCracken, & Guzdial, 2005). A set of pre- and post-test multiple choice question (MCQ) instruments was developed to evaluate students understanding of common CS1 topics, adapted from a similar approach used in an ITiCSE working group (Lister et al., 2004).

Participants from an engineering CS1 course and participants from a traditional CS1 course for majors had similar overall scores on the pre-test, with students, on average, answering 42.3% of questions about the introductory material correctly. However, on the pre-test there were statistically significant differences in understanding on 5 concepts (See Table 1 for details). Students who had completed the engineering introductory course demonstrated significantly better understanding of the iteration and conditional topics while students in the computing group demonstrated significantly better understandings of the binary search tree, array, and sorting topics.

A comparable[1] MCQ assessment was given to students at the end of a common second computing course. Students demonstrated improved understanding on almost

[1] *Comparable* forms of assessment are very similar in content, but the statistical similarly has not yet been proven (American Educational Research Association, American Psychological Association, & National Council on Measurement in Education, 1999).

all of the topics when compared to their pre-test levels, with an average score of 61.53%. Analysis of the post-test results revealed that after the common second course there was no longer a significant difference. Student understanding had converged and their performance was no longer distinguishable based upon their CS1 course.

The results of this study suggest that there are detectable differences in student understanding of introductory computer science concepts when students complete a first course using different pedagogical approaches. However, repeating the study in subsequent semesters with comparable student populations and course content that remained essentially unchanged did not return similar results.

So, despite using best practices in computing education today, of locally developed measures augmented by content validity review, a plausible explanation is that the measurement instrument themselves are flawed. 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 curriculum. 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 confirm student mastery of course concepts and completion of learning objectives. Accurate measures permit direct investigation of curricular innovations to evaluate whether or not the changes produced the intended outcomes. Measurement might also allow for the comparison of different pedagogical approaches enabling faculty to make decisions about curriculum and pedagogy informed by educational research. Computing education and research suffer from the

lack of such instruments.

The aim then is to have valid ways of measuring student conceptions of fundamental computer science topics, which will enable both research into how understanding of knowledge in our domain develops as well as enable curricular innovation and reform grounded in this knowledge. My dissertation will explore the development of an assessment instrument for introductory computer science concepts that is applicable across a variety of current pedagogies and programming languages.

1.1 Thesis Statement

It is possible to construct an assessment of fundamental computer science concepts that

(a) is widely applicable across a wide variety of current pedagogical approaches and paradigms;

(b) tests conceptual knowledge independently of the programming language used in the students' CS1 course; and

(c) is a valid measure of students' knowledge as demonstrated through think aloud interviews, empirical analysis of assessment results, and correlation with CS1 exam scores.

1.2 Research Questions

To address this thesis, I pose three broad research questions, which will be investigated across six studies. Table 2 summarizes the research questions and the corresponding studies that address each question. I describe the questions in the remaining sections of this chapter.

Table 2: Summary of Research Questions & Studies

Research Question	**Study**
RQ1: How can existing test development methods be applied and adapted to create a valid assessment instrument for CS1 conceptual knowledge?	
RQ1.1: How can a framework of common CS1 conceptual content be defined?	S1: Document analysis study of CS1 textbooks
RQ1.2: To what extent can validity of the assessment instrument be demonstrated by correlating with other valid instruments testing related content?	S2: Pilot validity study
RQ2: To what extent can pseudo-code be used as the mechanism for achieving programming language independence in an assessment instrument?	
RQ2.1: Is students' demonstration of fundamental CS1 conceptual knowledge, in closed-book examination questions, differentiated by programming language of instruction?	S3: Open-ended question study
RQ2.2: To what extent are students able to demonstrate their understanding of fundamental CS1 concepts in a pseudo-code assessment instrument?	S4: Think aloud interview study
RQ2.3: To what extent are students able to transfer their understanding of fundamental CS1 concepts from their introductory programming language of instruction to pseudo-code?	S5: FCS1 Assessment study
RQ3: To what extent does the language independent instrument provide a valid measure of CS1 conceptual knowledge?	S6: Validity study

1.2.1 Method for Developing a Validated CS1 Assessment Instrument

The fields of education and psychology have developed a rich history in developing and validating measurement instruments for a variety of purposes (Lindquist, 1951; American Educational Research Association et al., 1999). However, the field of computing does not currently have any validated assessment instruments of CS1 conceptual knowledge for pedagogical or research purposes. I applied these established methods and practices for developing valid measures, adapting them where necessary for the field of computer science.

RQ1: *How can existing test development methods be applied and adapted to create a valid assessment instrument for CS1 conceptual knowledge?*

Traditional test development follows an iterative process beginning with specification and verification of the content and purpose of the exam. A test bank of questions is then created and refined through a series of pilot studies. After the final candidate questions have been selected, empirical studies are used to establish the validity and reliability of the exam.

To create an assessment instrument for CS1, the Foundational CS1 (FCS1) Assessment, I adopted the methods from educational and psychological test development with two proposed adaptations. The first methodological change centered around creating an exam focused on concepts not programming language syntax, so the assessment can be as widely applicable as possible. The method requires the addition of a step to verify the programming language independence of the exam and to ensure that students are able to demonstrate their understanding adequately in the new language independent exam.

The second change is required because the standard methods for validating the instrument against existing valid measures do not apply to this exam, which will be the first of its kind in the field of computing. So the validity argument was crafted using a combination of think aloud interviews and statistical analysis techniques. In

sum, the data provided sufficient evidence that the assessment was indeed measuring the intended constructs.

RQ1.1: *How can a framework of common CS1 conceptual content be defined?*

One of the first steps in developing a validated assessment instrument is to specify the particular topics in a domain or area that are to be covered by the exam. For this research question, I explored ways to identify a framework of foundational computing concepts that CS1 courses, regardless of programming language or pedagogical paradigm, have in common. I conducted a document analysis of widely adopted CS1 textbooks, as an external representation of course content, while also being guided by the current ACM/IEEE curriculum guidelines (The Joint Task Force on Computing Curricula, 2001). The findings outlined a set of foundational CS1 concepts that were common across a wide variety of current pedagogical approaches and paradigms. Further these concepts were recognized and validated by experts to be representative of computing knowledge in an introductory computer science course.

Most CS1 courses share a set of learning goals for students to understand basic computing concepts and to be able to demonstrate use of those concepts in the execution of programming skill. For the FCS1 Assessment, I focused on the identification and measurement of common concepts. This learning goal is more amenable to standardized testing formats and to establishing the reliability and validity of the resulting instrument. In addition, there are wide variations in the skill level of programmers (Sackman, Erikson, & Grant, 1968), and developing a test to accurately measure that ability would suffer from the enormous disparity.

RQ1.2: *To what extent can validity of the assessment instrument be demonstrated by correlating with other valid instruments testing related content?*

Normally new assessment instruments are validated by correlating participants' scores on the new measure with their score on an existing validated measure of the same or very similar content (American Educational Research Association et al.,

1999). However, since the field of computing does not have such measures, this research question explored the feasibility of collecting validity evidence through a correlation study. Specifically, I conducted a pilot validity study with a version of the FCS1 Assessment, rewritten in the Java programming language, and the MCQ portion of the 2004 CS Subject Advanced Placement exam. I investigated whether participant scores on these two exams, with my version of the assessment specifically constructed to be as close a match as possible, showed evidence of a positive correlation. However, the results demonstrated that the content and/or purpose of existing validated computer science assessment measures are too dissimilar to the goals and content of the FCS1 Assessment. Thus existing measures of CS knowledge will not serve as useful tools in providing validation evidence.

1.2.2 Programming Language Independence

The goal of the FCS1 Assessment instrument is to have a widely applicable measure of fundamental CS1 concepts that is unbiased by any particular programming language. The broad research question examined was as follows:

RQ2: *To what extent can pseudo-code be used as the mechanism for achieving programming language independence in an assessment instrument?*

My aim is to develop a programming language independent assessment instrument, using pseudo-code as the method for achieving language independence, and I posit three questions to investigate the feasibility of this approach.

RQ2.1: *Is students' demonstration of fundamental CS1 conceptual knowledge, in closed-book examination questions, differentiated by programming language of instruction?*

The first step in investigating the feasibility of a programming language independent exam was to determine whether students' conceptions and misconceptions on the common CS1 concepts identified in Study 1 are significantly influenced by

the syntactic constructions of the programming language used in their introductory course. I gave students, who are completing a CS1 course in Java, Matlab, or Python, open-ended versions of the FCS1 Assessment questions. I examined the students' answers looking for patterns in the responses provided, particularly among the errors students make. Common incorrect responses across the programming language participant groups were identified, which suggested that students conceptual errors are similar regardless of their syntactic expression. These common errors provided a basis for the distractors in the multiple-choice version of the assessment instrument.

After the feasibility of a language independent exam was established, a pseudo-code for expressing the CS1 conceptual questions was used as the mechanism to achieve language independence. Two subsequent questions arise — are students able to demonstrate their conceptual understanding in this pseudo-code language and does the understanding students developed in their CS1 course transfer into this new approach?

RQ2.2: *To what extent are students able to demonstrate their understanding of fundamental CS1 concepts in a pseudo-code assessment instrument?*

I investigated this question by conducting think-aloud interviews with students completing the FCS1 Assessment. Participants were selected from introductory courses taught in three different programming languages (Java, Matlab, and Python) and across ability groupings (high, medium, low). I used qualitative analysis techniques on the interview transcripts to investigate students' ability to read and reason in pseudo-code. The majority (83.70%) of student responses demonstrated successful use of the pseudo-code syntax to read and reason about the relevant computing concepts.

RQ2.3: *To what extent are students able to transfer their understanding of fundamental CS1 concepts from their introductory programming language of instruction to pseudo-code?*

The final test of programming language independence is whether students were able to demonstrate comparable levels of conceptual understanding in pseudo-code as in their "native" programming language. To investigate this question, I ran a large scale empirical study comparing student performance on the FCS1 Assessment to a comparable version of the assessment instrument written in the students' CS1 programming language. Empirical evidence showed a strong positive correlation (Pearson's $r = .572$) between participant scores on the pseudo-code version and the language version of the assessment, and that the strong positive correlation held for each programming language participant sub-population. In addition, the findings show that students of higher ability levels demonstrated greater aptitude for transferring knowledge of CS1 concepts into pseudo-code.

1.2.3 Establishing Validity

After the assessment instrument has been piloted and revised, the final step in the development process was to establish the validity of the exam. Validity is a measure of "how well the test serves the purpose for which it is used" (Lindquist, 1951, 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.

RQ3: *To what extent does the language independent instrument provide a valid measure of CS1 conceptual knowledge?*

Given the position of this exam as the first of its kind in the field, I developed a three-pronged approach to establishing the validity of the assessment instrument. First, the think-aloud interviews allowed investigation into student reasoning while they were answering the questions on the exam. Analysis of these transcripts (Study

4) provided evidence whether students were answering questions based on their knowledge of the conceptual content. Secondly, statistical analysis techniques and item response theory (IRT) of responses to individual questions in the FCS1 Assessment Study (Study 5) provided evidence that the items are indeed measuring the desired constructs, rather than something more or less than intended. And lastly, I correlated students' scores on the assessment instrument with their exam scores in their CS1 course. Students' exam scores were used as a measure of their level of understanding of CS1 content as defined by external faculty teaching the course. By recruiting students from multiple institutions, I was able to mitigate the bias of correlating to a particular definition of the content of CS1.

The results and analysis show that the Foundational CS1 Assessment instrument does provide a valid measure of foundational CS1 content for procedurally-based CS1 courses taught in Java, Matlab, and Python. Further, there is a strong positive correlation between student CS1 final exam scores and the scores on the assessment.

1.3 Overview of Dissertation

The remainder of this document outlines the six studies that comprise my thesis work. Chapter 2 provides a review of relevant literature. Chapter 3 outlines the method for developing a validated language independent assessment of CS1 concepts and the associated studies used to evaluate the proposed method. Chapter 4 details the study designs used to evaluate the language independent nature of the assessment. Chapter 5 discusses the plans to determine the validity of the exam, and finally Chapter 6 provides a summary of the research contributions and a discussion of future work.

CHAPTER II

BACKGROUND AND RELATED WORK

This chapter begins with an overview of prior work and research in the field of assessment in the science, technology, engineering, and mathematics (STEM) community at large. I then then look at assessment efforts in computer science and differentiate my work from these existing attempts as a programming language independent assessment of conceptual knowledge for pedagogical and research use.

2.1 Assessment in STEM Disciplines

The STEM community has a rich tradition of educational research investigating teaching and learning across a wide variety of disciplines. Researchers in these fields have posited models of student development of knowledge, compared pedagogical approaches, and developed validated assessment tools and techniques to measure student understanding.

STEM assessment tools generally focus on either student attitudes or perceptions towards a discipline (e.g., Colorado Learning Attitudes about Science Survey (Adams et al., 2006)) or student mastery of the knowledge or skills in a discipline (e.g., Mechanics Baseline Test (Hestenes & Wells, 1992)). While attitude and motivation is a significant factor in student learning (Pintrich & Schunk, 2002), attention is focused on the concept-based assessments and how they might inform the design and development of the Foundational CS1 Assessment.

Researchers at the University of Wisconsin have created the Field Tested Learning Assessment Guide (FLAG) (College Level One Team, n.d.), an online resource for assessment across STEM disciplines. FLAG contains tested assessment instruments indexed by both the discipline and the technique (e.g., attitude survey, concept test,

MCQ), as well as articles guiding the adoption and adaptation of the techniques. Citations to relevant theory and study results are included for each instrument included in the repository. (The only instruments currently listed under Computer Science are general attitude surveys of science, textbooks, and writing. While these instruments certainly can be used in computing classes, they are not specific inquiries into the nature of the discipline.) Although a wide variety of disciplines have assessments included in the repository, from Agriculture to Engineering, a review of the research and development in assessment in two areas, Mathematics and Physics, is presented here as exemplars of the larger STEM community.

2.1.1 Mathematics

The mathematics education research community is perhaps the most well established disciplinary research community having been established in the 1960s as a collaboration between mathematicians, psychologists, and mathematics educators. In 1970 the first issue of the long-running Journal for Research in Mathematics Education was published by the National Council of Teachers of Mathematics (NCTM).

As a result of this history, the field of mathematics has a number of assessment instruments available to researchers and practitioners at the K-12 and university level. *Measuring Up* (Mathematical Sciences Education Board & National Research Council, 1993) provides a set of example assessment tools, and scoring rubrics, for fourth grade math students based upon the NCTM standards . The authors stress the importance of creating measures according to the standards and having clear grading criteria. Similar to the CS curriculum guidelines outlined in CC2001 (The Joint Task Force on Computing Curricula, 2001), the NCTM established a set of standards for assessment but did not prescribe or suggest any particular strategy for meeting those guidelines. The Balanced Assessment in Mathematics Program at the Harvard Graduate School of Education (Schwartz & Kenney, 2008) responded

by providing a specification of elementary and secondary mathematics and a set of assessment tasks and scoring rubrics that align with the original goals. Further, the mathematics community has embraced the challenges of aligning learning objectives, classroom activities, and assessment practices by providing professional development and training for teachers (Driscoll & Bryant, 1998).

The field of mathematics also has assessments designed for the collegiate level. The Basic Skills Diagnostic Test (Epstein, 1997) is an assessment of algebra knowledge and skills often used as a diagnostic of the level of preparedness for college level mathematics. The Calculus concept inventory is being developed to test basic principles of differential calculus – functions and derivatives. (Concept inventories are a specific form of concept-based assessment that will be discussed in greater detail in the following section, Section 2.2). Research studies using these assessment instruments showed that students did not perform as well as expected on pre-tests, nor did they show meaningful learning gains after a entire course covering these concepts (Epstein, 2006).

2.1.2 Physics

The Force Concept Inventory (FCI) (Hestenes et al., 1992) is the first concept inventory to be widely adopted and remains one of the most widely used and cited across all STEM disciplines. Hestenes, Halloun, and Wells designed the FCI to assess student understanding of the Newtonian concepts of force and were initially criticized for creating questions that were deemed too simple by many university faculty. However, studies showed that nearly every single student (n = 478) showed evidence of non-Newtonian thinking and that these misconceptions were resistant to instruction (Halloun & Hestenes, 1985b, 1985a).

> Measurements with the instrument show the student's initial qualitative, common sense beliefs about motion ... has a large effect on performance

> in physics, but conventional instruction induces only a small change in those beliefs. (Halloun & Hestenes, 1985b, p. 1043)

Harvard Physics professor, Richard Hake, contributed to the adoption of the FCI across the academic physics community when he gave the instrument to his students and was surprised to get similar results – students were reasoning using "common sense" theories of physics and these theories persisted after a semester of university physics. Hake then began an investigation into whether there were pedagogical techniques that would lead to meaningful conceptual knowledge gains (Hake, 1998). Data was collected from students in 62 physics courses across the country (n = 6542). Analysis showed that "interactive engagement" techniques, usually involving hands on activities, resulted in nearly double the gain as traditional lecture-based methods. These findings have led to a research agenda in the physics education research community around active learning and student engagement (Crouch & Mazur, 2001).

Due to the success of the FCI, many other areas and disciplines have created similar instruments. Examples of other concept inventories in use across STEM disciplines include:

- Brief Electricity & Magnetism Assessment (BEMA) (Ding, Chabay, Sherwood, & Beichner, 2006)
- Light and Spectroscopy Concept Inventory (LSCI) (Prather, Rudolph, Brissenden, & Schlingman, 2009)
- Genetics Concept Assessment (GCA) (Smith, Wood, & Knight, 2008)
- Geoscience Concept Inventory (Libarkin & Anderson, 2005)

2.2 Concept Inventories

A concept inventory is a particular form of concept-based assessment, specifically designed to probe a person's understanding of a specific set of narrowly defined concepts.

For example, the FCI was not designed to explore students' knowledge of introductory physics, but rather just their understanding of forces. Concept inventories use a multiple-choice question (MCQ) format for objective scoring, but the formulation of both the question stem and distractors are a result of extensive research based on models of concept understanding and misconceptions surrounding that concept. The resulting concept questions present both the correct answer as well as distractors formulated from commonly held misconceptions.

The research and design of the assessment is explicitly built around elicitation of students' ways of thinking and expressing their understanding rather than relying on expert opinion or assumption. As a result, student responses to a concept inventory inform understanding about student thinking and provide researchers and teachers with evidence to the ideas, misconceptions, and knowledge gaps that students bring to a classroom. Again using the FCI as the example, researchers began with a preliminary mechanics diagnostic test, using short answer, open-ended questions, given to more than 1000 introductory physics students over three years (Halloun & Hestenes, 1985b). A taxonomy of common sense concepts that conflict with Newtonian theory was compiled from this data (Halloun & Hestenes, 1985a), which was then used to create a detailed model of instruction and student learning for introductory physics (Hestenes, 1987). This model of student knowledge development and the corpus of misconceptions were used to develop the FCI.

Unlike natural sciences, computer science knowledge has no real world analog. Thus it is likely that a majority of student misconceptions are a result of instruction in CS rather than based upon a set of common, naive understandings they bring to the topic from their experience in the world. Further, as a relatively new disciplinary educational research community, we do not know how students come to learn about CS or introductory programming topics, know what topics should be pre-requisites to others, or have a model of development of computing knowledge. Due to these

limitations, I chose to adapt the model of a standardized, concept-based assessment instrument, relying on a framework of common conceptual content to define the scope of the concepts to be assessed.

2.3 Assessment Practices in Computing

Current assessment practices in computing have a number of limitations which ultimately restricts the availability of instruments for pedagogical or research purposes. There are two types of assessment efforts in computing – existing validated exams which are developed, owned, and administered by national/international examining boards and CS education research projects. Program assessment of computer science programs, usually done for accreditation purposes (Rogers, n.d.), is a peer-review process focused on institutional assessment of student learning. Due to the focus on high-level programmatic objectives, I exclude these types of assessments from further discussion.

Two validated instruments focus on the introductory sequence in computing. The CS Advanced Placement exam is one of the most popular validated assessment instruments in computing. This high school exam, usually taken in the junior or senior year, is typically used to earn credit for a CS1 course in college. The AP exam is currently written in the Java programming language and has been criticized for relying too heavily on syntactic level details rather than programming concepts. The United Kingdom also offers a high school level exam, the Advanced Level General Certificate of Education (A-level) in Computing. The goal of this assessment is to demonstrate mastery of course content and the instrument uses a short answer and practice problem format. This format requires extensive training and workload for examiners to achieve reliable scoring.

The remaining two validated instruments are designed for students completing their computer science degree programs. The Major Field Test for Computer Science

is designed to measure student progress and evaluate end of program outcomes. Designed to assess mastery of an entire degree program, the scope of the exam is fairly broad, including systems, theory and programming. The GRE Computer Science Subject Test, also taken by students completing their degree, is designed to predict success in graduate school. The exam is similarly scoped and includes software systems, computer organization, and theory.

The CS education community has shown growing interest in assessment research, and two related projects are underway. In dissertation work, Decker (2007) designed an assessment for an introductory sequence of programming courses (CS1 and CS2) in the Java programming language. The short-answer format instrument was developed and tested at a single institution, and therefore its validity claims cannot be generalized beyond that context.

Craig Zilles and colleagues have received funding (NSF-CCLI #0618589) to develop a set of concept inventories for computing, in discrete math, digital logic design, and programming fundamentals. The researchers have elicited the set of troublesome concepts from educational experts (Goldman et al., 2008) and are conducting think-aloud interviews to capture student misconceptions (Herman, Kaczmarczyk, Loui, & Zilles, 2008; Kaczmarczyk, Petrick, East, & Herman, 2010). The work is preliminary and the instruments are still being developed. However, progress reports (Herman, Loui, & Zilles, 2010) suggest that the process and resulting assessment are going to be much more closely aligned with a traditional standardized assessment instrument than a concept inventory.

The existing assessment instruments in computing suffer from one or more of the following issues. If an assessment instrument is tied to a particular programming language, its applicability is limited since it cannot be used across different approaches or courses using other programming languages. Similarly, some instruments focus on syntactic programming language details, rather than the higher order concepts that

are more aligned with traditional course learning objectives. Finally, some tests have psychometric or predictive aims, so do not focus specifically on assessing learning and should not be used for those purposes.

My work differs from these efforts in that I aim to create a rigorously validated exam that could be widely adopted and used in any introductory CS course. The 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.

In this chapter I have motivated the need to create a validated, language independent assessment of introductory programming concepts. I described current approaches to assessment in the STEM education, and explained how current assessment efforts in the CS education research community are not sufficient to meet the needs of the research or practitioner communities.

The chapter that follows outlines the method I used to develop the Foundational CS1 Assessment. Chapter 3 also describes the research questions and studies associated with adapting and applying the general purpose educational methods of assessment research and development into a specific disciplinary context.

CHAPTER III

METHOD FOR DEVELOPING A VALIDATED CS1 ASSESSMENT INSTRUMENT

Psychology and education have a strong tradition of creating validated assessments and exams for a wide variety of purposes (Lindquist, 1951; American Educational Research Association et al., 1999). Since computing does not currently have any of these instruments that are applicable for educational research, I applied these established methods and practices for developing valid measures, adapting them where necessary for the field of computer science (Tew & Guzdial, 2010).

3.1 Adapting Standard Assessment Development Practices

As introduced in Chapter 1, the primary research question here was methodological.

RQ1: *How can existing test development methods be applied and adapted to create a valid assessment instrument for CS1 conceptual knowledge?*

The method designed is based upon standard educational test development guidelines, summarized below (American Educational Research Association et al., 1999). 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 is 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. Piloting of the questions then takes place. Pilot tests 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 examinations, and thus are dependable.

To create the Foundational CS1 Assessment, two adaptations were required. The first methodological change centered around creating an exam focused on concepts not programming language syntax, so the assessment can be as widely applicable as possible. The method required the addition of a step to verify the programming language independence of the exam. To achieve language independence for a CS1 exam, I utilized a verbose pseudo-code as the exam programming language. I evaluated the effectiveness of this approach using a combination of think aloud and pilot studies. (See Chapter 4 for more details). These studies were required to ensure students are able to appropriately transfer understanding from their programming language of instruction to pseudo-code and to ensure that students are able to demonstrate their understanding adequately in the new language independent exam.

The second change is required because the standard methods for validating the instrument against existing valid measures did not apply to this exam, which is the first of its kind in the field of computing. So a validity argument was crafted using a combination of think aloud interviews and statistical analysis techniques. In sum, the data should provide sufficient evidence that the assessment is indeed measuring the intended constructs. (See Chapter 5 for more details).

In order to achieve language independence, I augmented the standard development 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 three steps in this process will be discussed in the remaining sections of this chapter. The research questions and studies associated with steps 4 and 5 are explained in Chapter 4. The process of establishing validity (step 6) is addressed in Chapter 5. The data collected will allow preliminary analysis of the internal reliability of the exam. However, full-scale reliability testing (step 7) is beyond the scope of this dissertation.

3.2 Defining Conceptual Content

Given the goal of developing a widely applicable CS1 assessment, the strategy for defining content was to identify concepts that a wide variety of introductory courses and approaches had in common. The research question and hypotheses addressed by this study were:

RQ1.1: *How can a framework of common CS1 conceptual content be defined?*

H1: A set of concepts that are amenable to testing across CS1 languages and paradigms can be identified.

H2: The concepts will be recognized by a panel of experts to be representative of foundational CS1 knowledge.

3.2.1 Study Method

To identify a set of common conceptual content, I conducted a document analysis of both introductory textbooks and CS curriculum guidelines. The document analysis had four phases: an analysis of the table of contents of widely adopted CS1 textbooks; scoping using current ACM/IEEE CS curriculum guidelines (The Joint Task Force on Computing Curricula, 2001); refinement using canonical introductory textbooks representative of common pedagogical approaches; and synthesis using a thematic analysis to identify the final list of concepts. This iterative approach allowed the concepts to be derived from resources that emphasized both top-down and bottom-up approaches to specifying course content. Details of the analysis are provided in the following sections.

3.2.2 Data Analysis

I began by conducting a document analysis of the table of contents of the two most widely adopted CS1 textbooks as identified by 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 (Cohoon & Davidson, 2006; Deitel & Deitel, 2005; Horstmann, 2005, 2006; Lewis & Loftus, 2005; Malik, 2004, 2006; Mercer, 2002; Savitch, 2005a, 2005b; Wu, 2006; Zelle, 2004).

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 the 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).

I used the framework of the Computer Science volume of Computing Curricula 2001 (The Joint Task Force on Computing Curricula, 2001) to revise the 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. I eliminated any concepts outside of the scope of the introductory sequence and further narrowed the intended scope of the 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. The next phase in my analysis was to focus on canonical texts representing each of the common introductory approaches (objects-first (Lewis & Loftus, 2005; Deitel & Deitel, 2005), functional-first (Felleisen, Findler, Flatt, & Krishnamurthi, 2001), and imperative-first (Zelle, 2004)). By conducting a document analysis of these texts, I was able to focus the topic list on the set of concepts that are included by a variety of CS1 approaches, but avoided dilution by including too many data points in the process. 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 3.

Table 3: 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

The topics in Table 3 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. I therefore performed additional thematic analysis (Braun & Clarke, 2006) and synthesis with the aim of generating a small handful of constructs that were 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, string) provide useful information for the kinds of data commonly available for manipulation in test questions, but I 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, the object construct was limited to the basics of class definitions and method calls.

The final list of constructs which serve as the basis of the test specification for the FCS1 Assessment are as follows:

- Fundamentals (variables, assignment, mathematical expressions)
- Logical Operators
- Selection Statement (if/else)
- Definite Loops (for)
- Indefinite Loops (while)
- Arrays
- Function/method parameters

- Function/method return values
- Recursion
- Object-oriented Basics[1] (class definition, method calls)

3.3 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 (American Educational Research Association et al., 1999).

For the FCS1 Assessment, each construct is weighted equally with 10% of the questions devoted to each topic, and the questions are in a multiple-choice question (MCQ) 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 (Haladyna, 2004; Lukhele, Thissen, & Wainer, 1994). 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 the 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

[1]Due to the distinct syntax mechanisms different programming languages use to express object-oriented notation, I opted not to include questions about object-orientation on the first iteration of the FCS1 Assessment.

Q12.

Consider the following code segment.

```
x = 0
y = 1

x = 3 * y
```

During code execution, which of the following statements are always true?

I. `x` is a declared variable.
II. `y` is a declared variable.
III. The value of `x` depends on the value of `y`.

A. I only

B. III only

C. I and II

D. I and III

E. I, II, and III

Figure 1: Example Definitional Question

Consider the following code segment.

```
flag1 =  False AND (True OR False)
flag2 = (False AND True) OR (False AND False)
flag3 = (False OR True) OR False
flag1 = (flag1 OR (NOT flag2)) AND flag3
```

After the above code is executed, which of the following statements are true?

```
I.    flag1 == True
II.   flag2 == True
III.  flag3 == True
```

A. I only

B. II only

C. III only

D. I and III

E. I, II, and III

Figure 2: Example Tracing Question

types of questions about each construct were developed. *Definitional* questions explore the student's general understanding of a construct. A sample definitional question is shown in Figure 1[2]. (A more detailed analysis and discussion of this question, Q12, appears in Section 4.3.) *Tracing* questions examine a student's ability to predict execution of code using a particular concept (e.g. the value of a variable after a loop completes execution). *Code completion* questions are fill-in-the blank type questions to evaluate a student's ability to write code. See Figures 2 and 3 for an example of each. For each construct, I built multiple versions of each type of question for the test bank to allow for the evaluation and selection of the best questions. Test construction, followed established heuristics for writing multiple-choice questions (Miller, Linn, & Gronlund, 2009a). Some of the guidelines are as follows:

- The item stem should include as much information as possible and should avoid irrelevant material.
- An item should contain only one correct or clearly best answer.
- All distractors should be plausible.
- The relative length of the alternatives should not provide a clue to the correct answer.
- The correct answer should appear in each of the alternative positions an equal number of times but in random order.
- Use sparingly special alternatives such as "None" or "All of the Above".

[2]The example questions included are representational of the kinds of questions that were asked on the FCS1 Assessment. However, publication is only possible because they have been discarded from the test bank after pilot testing.

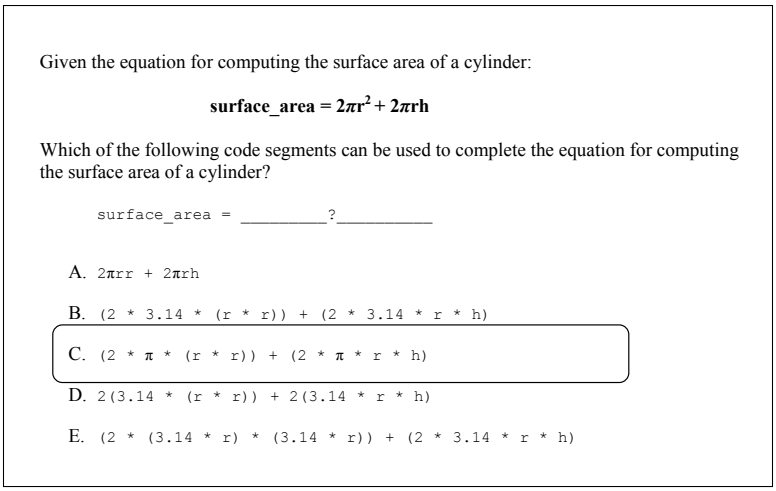


Figure 3: Example Code Completion Question

3.3.1 Study 1 Findings and Contributions

- A set of foundational CS1 concepts that are common across a wide variety of current pedagogical approaches and paradigms. (H1, confirmed)
- While not an exhaustive list, the specification of concepts was validated by experts to be representative of foundational CS1 knowledge. (H2, confirmed)

3.4 Pilot Validity Study

Validity of a new assessment is normally established by correlating participant scores on the new instrument with scores on an existing test of similar content that has already been validated (American Educational Research Association et al., 1999). The validity argument is as follows – if the scores on the new instrument correlate with scores on a previous instrument that has been shown to accurately measure the concepts, then the test must also accurately measure the intended concepts. However, since computer science does not have an existing validated instrument that measures the same content, I studied the feasibility of collecting validity evidence through a correlation study using a best-case scenario. The research questions and hypothesis investigated in this study were:

RQ1.2: *To what extent can validity of the assessment instrument be demonstrated by correlating with other valid instruments testing related content?*

H3: The content and/or purpose of existing validated computer science assessment measures are too dissimilar to the CS1 assessment instrument to be useful tools in establishing validity.

Specifically, I conducted a pilot validity study with a version of the FCS1 Assessment, rewritten in the Java programming language, and the MCQ portion of the 2004 CS Subject Advanced Placement exam. I investigated whether participant scores on these two exams, with my version of the assessment specifically constructed to be as close a match as possible, showed evidence of a positive correlation.

3.4.1 Participants and Recruitment

Participants were recruited from high school computer science advanced placement (AP) courses in the greater Atlanta area during Spring Semester 2009. AP computer science teachers were asked to provide class time for participation and encouraged to use the study instruments as practice exams before the students took the AP exam in May. This population was selected because data could be collected under testing conditions as the students were motivated to prepare for the upcoming AP exam. Further the content of the CS AP curriculum is standardized and published which allows for direct comparison of the topics covered by the CS1 assessment. Students and their parents were provided informational fliers about the study and then signed assent and consent forms if they agreed to participate. Seven high schools, both public and private, participated yielding a total of 63 student participants.

3.4.2 Study Method

The study[3] consisted of two assessment exams given under testing conditions – no questions were permitted and collaboration was not allowed. Teachers were asked to administer the tests at least one week apart and to provide the same amount of time to complete each test. To ensure adequate AP test preparation, the high school teachers first gave the 40 question MCQ portion of the 2004 AP Exam. In a subsequent class period, students were given the 27 question CS concept assessment written in Java. Participant scores were coded by a unique identifier so their scores on the two exams could be correlated.

3.4.3 Data Analysis

Data analysis occurred in two stages: the first stage searched for correlations in the data set between participant scores on the two assessment exams, and the second stage probed the FCS1 Assessment assessment data in greater to detail to seek evidence of correlations by concept area.

Before the initial analysis could be completed, the AP data set was filtered. Five questions pertained to a case study for which current students had not prepared, since they were preparing for the current year's case study. There were also thirteen questions on software engineering and object-oriented programming, which were not covered by the CS1 assessment. Data for these questions were removed resulting in a 22 question data set.

A Pearson's product moment correlation coefficient was computed to assess the relationship between the participant score on the AP exam and participant score on the FCS1 Assessment. Overall, there was a weak positive correlation between the two variables, (Pearson's $r(63) = 0.159$, $p = 0.215$.) Subsequent analysis focused on whether there were stronger correlations when participant scores by particular

[3]Institutional Review Board approval was obtained for all protocols involving human subjects.

Table 4: Pilot Validity Study Correlation Coefficients by Concept

Concept	***r***	***p***
Fundamentals	-.048	.707
Logical Operators	-.073	.567
Selection Statement	-.019	.882
Definite Loops	.051	.691
Indefinite Loops	-.114	.376
Arrays	.040	.754
Recursion	.197	.122

concept areas were examined, rather than the overall score. The AP exam questions are classified into categories mapping to seven of the concepts previously identified in Section 3.2: basics, logical operators, selection statements, definite loops, indefinite loops, arrays and recursion. Pearson correlation coefficients were computed for each of these topics and resulted in topic correlations ranging from -11% – 20%. (See Table 4 for the details of each correlation.) While recursion exhibited the strongest correlation, no finding was significant.

The results from our data analysis have a number of implications. First, the Java focus of the AP exam may skew the evaluation of some of the questions towards programming language details rather than semantic concepts. For example, there are questions that investigate runtime and/or compilation errors, and there are questions that examine the correct syntax for method calls. Further some of the AP questions are explicitly designed to cover multiple concepts, so it is not feasible to tease out individual concept mappings. The weak correlations suggest that existing valid instruments are not going to prove useful in constructing the validity argument for the FCS1 Assessment instrument. Even in the best case scenario, with the assessment rewritten in the Java programming language and extraneous questions removed from the AP exam data set, there were no significant or strong correlations.

3.4.4 Study 2 Findings and Contributions

- Existing validated exams in CS have test specifications that are sufficiently different than the FCS1 Assessment in either content, format or objectives.
- Correlation studies, using existing validated CS instruments, will not provide useful evidence for establishing validity of the FCS1 Assessment. (H3, confirmed)

In this chapter I have designed a method for developing the FCS1 Assessment, specified a set of common foundational CS1 concepts, and piloted a method for establishing validity of the new exam. In so doing, I have addressed research questions RQ1, 1.1, and 1.2 and presented evidence confirming hypothesis H1, H2, and H3. The next chapter discusses the research questions and studies used to verify the programming language independence of the FCS1 Assessment.

CHAPTER IV

VERIFYING PROGRAMMING LANGUAGE INDEPENDENCE

The goal of the Foundational CS1 Assessment is to have a widely applicable measure of introductory CS1 concepts that is unbiased by any particular programming language. So I developed a programming language independent assessment instrument, and this chapter will discuss the questions I posed to investigate this approach. The broad research question examined here is as follows:

RQ2: *To what extent can pseudo-code be used as the mechanism for achieving programming language independence in an assessment instrument?*

The first step in exploring the possibility of a programming language independent exam was to determine whether students' conceptions and misconceptions on the CS1 concepts included in the FCS1 Assessment were significantly influenced by the syntactic constructions of the programming language used in their introductory course. After the feasibility of a language independent exam had been established, I explored using pseudo-code as the language to express the CS1 conceptual questions. Two subsequent questions arose - Are students able to demonstrate their conceptual understanding in this pseudo-code language? And are students able to demonstrate comparable levels of understanding in this pseudo-code and the programming language used in their introductory course?

4.1 Open-ended FCS1 Questions Study

There are many different programming languages used to teach CS1 today, and some of the most popular include C++, Java, Python, and Scheme. While Study 1 demonstrated that there were common semantic concepts that could be identified across these approaches, this research explores the degree to which novices express their understanding in common patterns. Is novices' knowledge representation, both correct and incorrect, so deeply rooted in syntax such that the errors are localized to a particular programming language? Or are the mistakes identifiable across the population of novices, regardless of their introductory programming language? The research question and hypothesis being addressed by this study were:

RQ2.1: *Is students' demonstration of fundamental CS1 conceptual knowledge, in closed-book examination questions, differentiated by programming language of instruction?*

H4: Trends and common conceptual responses across the introductory programming languages and paradigms will be identified among the student responses.

4.1.1 Participants and Recruitment

Participants, from four different universities, were recruited as they were completing their first course in computer science. The courses were taught in Java, Matlab, and Python. Open-ended versions of the FCS1 Assessment questions were given to students in scheduled examination sessions, as a portion of their regular course. While the examinations were a required portion of the students' course grade, their data was only included in the study if they volunteered to participate. There were a total of 304 participants who were enrolled in a CS1 in Matlab (n = 125), Python (n = 76), or Java (n = 103).

Table 5: Summary of Participants in Open-Ended FCS1 Questions Study

College	Programming Language	Testing Condition	n
U1	Matlab	Extra-credit	125
U1	Python	Extra-credit	76
U2	Java	Recitation Quiz	33
U3	Java	Pop Quiz	70

4.1.2 Study Method

In cooperation with CS1 faculty at each institution, open-ended versions of the FCS1 Assessment questions were placed into examinations. Faculty were asked to give the questions under testing conditions but were allowed flexibility as how to incorporate them. Two chose to use them as extra-credit questions on exams, one chose to use them as in-class pop-quiz questions, and one chose to use them as practice quizzes given in recitation before an upcoming exam. Student answers were collected anonymously, although the responses were coded by institution to capture the programming language and testing context. A summary of data collection appears in Table 5.

4.1.3 Data Analysis

Student answers were analyzed, looking for patterns in the responses provided, particularly among the errors students made. These common incorrect answers provided a basis for the distractors in the final draft of the multiple-choice version of the FCS1 Assessment. For the purpose of data analysis, the questions were divided into two types. *Closed-form* questions had clearly identifiable, concise answers (e.g., What is the value of the variable 'i' after loop execution?). *Short answer* questions were those that required students to provide brief explanations or to write portions of code. Seven questions on the test were of the closed-form and were analyzed using statistical techniques. Qualitative analysis techniques were used to analyze data from the remaining 20 short answer questions.

Student responses were first aggregated looking for common responses within the three programming language populations, Java, Matlab, and Python. Syntactic errors

were ignored, and an answer was only recorded into the dataset when it was identified by more than one participant. After common answers were identified, responses were compared across the populations. Exemplars from data analysis are presented here. (See Appendix B for details on the data collected in Study 3.)

4.1.3.1 Closed-form Exemplar

Question 8, an example of a closed-form question, asked students to trace the values stored in 3 integer variables (x, y, and z) through a set of nested selection (if) statements. The participants were able to easily trace the values of x and y with over 80% of the students providing correct responses. However, the variable z proved much more difficult. Only 40.7% of the participants were able to correctly trace its execution. The common incorrect answers identified for each participant group, ranked in order of its frequency of occurrence, are shown in Table 6. There were no common incorrect answers in the Java participant group for the y variable, so the analysis focuses on the remaining variables. While few participants gave incorrect answers to the value of x, the error of printing the value of $x = 25$ was made by all participant groups. However, since the majority of errors were made tracing the final variable, those results may be a better overall predictor of student understanding. The most common answer overall and in each participant group was $z = 49$.

Students showed very little desk-work or interim variable values, choosing merely to provide the final answer at the bottom of the page. So, even in open-ended form, these questions and analysis provide little insight into why the errors were made. However, this is initial evidence that students with training in different introductory programming languages do answer closed-form questions similarly, and that there are common errors made across the languages for these types of questions.

The majority of the questions on the FCS1 Assessment are of the short answer form and the data for these questions cannot be meaningfully analyzed using statistical

Table 6: Common Incorrect Answers to Closed-Form Question Q8 Ranked by Frequency of Occurrence

variable: x, 85.63% correct						
	Java		**Matlab**		**Python**	
	Value	n	Value	n	Value	n
1st	$x = 25^{\ddagger}$	4	error/nothing	4	$x = 25^{\ddagger}$	5
2nd			$x = 25^{\ddagger}$	2		

variable: y, 86.23% correct						
	Java		**Matlab**		**Python**	
1st			$y = 2$	9	$y = 4$	3
2nd			error/nothing	4		

variable: z, 47.90% correct						
	Java		**Matlab**		**Python**	
1st	$z = 49^{\ddagger}$	24	$z = 49^{\ddagger}$	32	$z = 49^{\ddagger}$	24
2nd			error/nothing	4	$z = 3$	2
3rd					$z = 14$	2

‡ Trend identified across all three languages.
* Trend identified across two languages, but answer appeared in all groups.
$^{\diamond}$ Trend identified across two languages.

techniques. The short answer questions can be further subdivided into two sub-types. *Explanatory* questions ask students to define or describe a concept or to summarize the execution of a short piece of code. *Coding* questions require students to write lines of code to create a method that satisfies a stated condition (e.g., write code to fill in the blanks to complete an acronym function).

Qualitative data analysis of the aggregated student responses proceeded in an iterative fashion. In an effort to focus on semantic constructs, the first pass through the data set was to correct for any obvious syntactic errors (e.g. omitting the return/newline at the end of a print statement). Student responses that were identical, except for these small syntactic errors, were sorted into the same response category. Student responses were then coded by the error that was made, allowing comparison of responses across programming languages. Other trends can be identified when the errors are analyzed in the context of the semantic concept. The individual programming languages enable the students to express errors in slightly different syntactic

Table 7: Common Incorrect Answers to Short Answer Question Q11 Ranked by Frequency of Occurrence

	Java	**Matlab**	**Python**
1st	REVERSE!‡	REVERSE!‡	REVERSE!‡
2nd	E! E! E! E! E! E! E!	E‡ SE RSE ERSE VERSE EVERSE REVERSE	E‡ SE RSE ERSE VERSE EVERSE REVERSE
3rd	E‡ SE RSE ERSE VERSE EVERSE REVERSE!	E!	E!
4th	E!◇ ES! ESR! ESRE! ESREV! ESREVE! ESREVER!	!	!
5th	E! S! R! E! V! E! R!		E!◇ ES! ESR! ESRE! ESREV! ESREVE! ESREVER!

‡ Trend identified across all three languages.
* Trend identified across two languages, but answer appeared in all groups.
◇ Trend identified across two languages.

ways. The final stage of data analysis will identify these trends, independent of how the error is expressed.

4.1.3.2 Short Answer Exemplar

Question 11 was an explanatory question that asked students to predict the outcome of a segment of code focused on nested for loops. The nested loops printed the characters of the array in reverse order, one step at a time. Students struggled to correctly predict the output, with only 26% doing so. Students provided a wide

variety of responses to the question. The top five common incorrect answers from each participant group are shown in Table 7. The most common error "REVERSE!", most likely made by following intuition about the purpose of the function rather than carefully tracing the nested loops, was made by all participant groups. The Java and Python participant groups share a common error of incorrectly computing the array index (the 4th and 5th most common errors respectively.) Abstracting away from syntactic expression, a common error identified in the last stage of analysis was incorrectly placing the command to print the "!" character outside of the scope of the nested loops (Java error 3, Matlab error 2, Python error 2).

The variety of responses to the open-ended question does limit the number of patterns that could be identified across the answers. But there is evidence that students are providing similar conceptual answers, even when their syntactic expressions may be different. The goal of this study and analysis is not to evaluate all student errors, but to identify a set of common conceptual errors that can be used as plausible distractors in the FCS1 Assessment. Results from data analysis on the remaining questions is presented in Appendix B.

4.1.4 Study 3 Findings and Contributions

- Evidence that common responses across the programming language participant groups can be identified from student data. (H4, confirmed)

4.2 Pseudo-code Design

I proposed using pseudo-code as the mechanism to express the FCS1 Assessment conceptual questions in a programming-language independent manner. The pseudo-code uses a very verbose style adapted from guides for beginning programmers published by Whitfort (n.d.) and Shackelford (1997). To help students learn the new language, syntax is kept to a minimum (e.g., no semi-colons or curly braces), reserve words are

capitalized, and program blocks are closed with specific end commands (i.e., END-FOR) (Sime, Green, & Guest, 1976). A overview of the constructs of the language was developed to include both the definition of the syntax as well as examples and sample code.

The design of the pseudo-code and the 2-page guide underwent formative evaluation by students enrolled in a variety of degree programs at Georgia Tech. Revisions were made to make the syntax more consistent and to clarify the guide. The definition of the pseudo-code language is included in Appendix C.

4.3 Think-Aloud Interview Study

After confirming the hypothesis of common student responses, I conducted two studies to investigate using pseudo-code to achieve programming language independence. The first of these studies explored students' ability to demonstrate their understanding of the FCS1 Assessment concepts in the new pseudo-code language. The research question under investigation was as follows:

RQ2.2: *To what extent are students able to demonstrate their understanding of fundamental CS1 concepts in a pseudo-code assessment instrument?*

H5: Students will be able to read and reason in pseudo-code.

H6: The majority of errors made by students will be conceptual, rather than syntactic, in nature.

4.3.1 Participants and Recruitment

Participants were recruited from introductory courses taught in three different programming languages (Java, Matlab, and Python) and across ability groupings (high, medium, low). Faculty were asked to divide their course roster into thirds based upon students' midterm averages. A listing of the students grouped by ability will be provided, but they will not be identified by the specific ability grouping to which they

Table 8: Summary of Participants in Think Aloud Study

Ability Bin	Java	Matlab	Python
1	J1, J3	M7	P1
2	J2, J4	M1, M3, M5	
3	J5	M4, M6	P2

belong. My goal was to capture a cross-section of student abilities in each programming language participant group and try to mitigate the bias of higher achievement levels often found in volunteers. The ability groupings are needed to ensure that students from a diverse set of skill levels are able to read and reason in the pseudo-code. I recruited 14 students, planning for some participant attrition, although only one participant (P2) failed to complete the study. See Table 8 for a summary of the participants according to their relative ability level in their introductory course.

4.3.2 Study Method

The study was a two-part think aloud interview conducted while participants were completing the FCS1 Assessment. Participants were given the overview of the pseudo-code and were given 5 minutes to familiarize themselves with the new syntax. They were then given the first 13 questions of the assessment and asked to think-aloud while completing the exam. The participants were given 1.5 hours to complete this portion of the exam. Participants then returned for a second think-aloud interview where they were asked to answer the remaining 14 questions in a similar 1.5 hour session. Participants were compensated at a rate of $10 an hour for their time. Audio recordings of the think-aloud interviews were made and coded according to the CS1 programming language of the participant and the ability bin (1, 2, 3).

4.3.3 Data Analysis

The interview data was transcribed and content analysis (Neuendorf, 2002) was used to analyze the participant responses. Specifically, I looked for correctness of the answers, errors that were made, and evidence of reasoning using the new pseudo-code

Table 9: Coding Rubric for Think Aloud Interview Data

Code	Description	%
1	Participant answered question correctly by reasoning about intended construct.	49.63%
2	Participant answered question incorrectly by following common misconception or using faulty logic about construct.	34.07%
3	Participant answered question correctly even though they had incorrect reasoning about construct.	5.92%
4	Participant answered question correctly, however the correct answer was reached by reasoning about other conceptual content.	0.00%
5	Participant answered question incorrectly due to reasoning about other constructs.	4.44%
6	Participant answered question incorrectly. The wording of the question led to confusion/incorrect answer.	5.18%
7	Participant answered question incorrectly. Difficulty with or inappropriate transfer from programming language to pseudo-code lead to confusion/incorrect answer.	0.74%
8	Participant answered question incorrectly. The reasoning was incoherent and difficult to assign to any particular concept/construct.	0.00%

language syntax. A coding rubric was developed to capture whether participants were answering the question correctly and what factors contributed to their reasoning. The rubric is presented in detail in Table 9.

The first two codes capture the scenarios of a participant reasoning about the question concept and answering the question correctly (code 1) or incorrectly (code 2). The third code recognizes situations in which the participant has a misconception or misunderstanding about the concept, but due to the multiple choice format of the exam, is still able to guess the answer correctly. The middle set of codes (4 & 5) mark situations where the participants' answer choice was driven by their reasoning about a construct other than the primary conceptual focus of the question. The last set of codes identify problematic areas with the exam. Code 6 designates a scenario in which the wording of the question causes confusion and leads to an incorrect answer. Code 7 is used to mark areas where the pseudo-code language caused difficulty, either in reasoning about the language itself or through inappropriate transfer from their CS1 programming language to pseudo-code. The last code (code 8) was designed to capture situations in which a participant's logic was incomprehensible, and it was

difficult to tie their reasoning to any particular conceptual construct.

The interview participants FCS1 Assessment exams were graded for correctness, and then study participants were randomly sampled from the data set making sure to have a high and low scoring participant from each programming language group. (Since only one participant in the Python group completed the think aloud interview study, participant P1 was automatically included in the data set.) A total of 5 participants were selected to have their interview data coded according to the scoring rubric. Responses to each question were coded independently by two researchers. Discrepancies in the coding were resolved collaboratively with a goal of clarifying the rubric's definition and ensuring a consistent application of the rubric across the entire data set. The results of the coding are presented in Table 10.

Table 10: Think Aloud Interview Rubric Scoring Data

Question	Answer	Concept	J3		M7		J4		M3		P1	
			Answer	Code	Answer	Code	Answer	Code	Answer	Code	Answer	Code
Q1	E	for	C	6	E	1	E	1	C	2	C	2
Q2	C	logical operator	C	3	C	1	C	1	E	2	E	2
Q3	C	while	C	1	C	1	E	2	A	2	C	1
Q4	E	arrays	E	1	E	1	B	5	E	1	B	2
Q5	A	function return values	A	1	A	1	B	2	E	2	D	2
Q6	A	if	A	1	A	1	A	1	A	1	C	6
Q7	C	while	C	1	C	1	C	1	D	5	C	1
Q8	E	for	E	1	E	1	E	1	E	3	D	2
Q9	D	while	D	1	D	1	D	1	E	2	D	1
Q10	C	logical operator	B	2	A	2	C	3	C	1	C	1
Q11	D	function return values	A	2	D	1	A	2	B	2	A	2
Q12	A	basics	C	6	E	6	C	6	C	2	E	6
Q13	A	for	A	1	D	2	B	2	B	2	B	2
Q14	E	recursion	E	1	B	6	E	1	B	2	E	1
Q15	E	function return values	A	5	E	1	A	2	A	2	A	5
Q16	D	function parameters	D	1	D	1	C	2	D	3	D	1
Q17	C	arrays	A	2	A	5	B	2	B	2	A	5
Q18	B	recursion	C	2	D	2	B	3	A	2	A	2
Q19	B	if	B	1	C	2	A	2	C	2	C	2
Q20	B	function parameters	B	1	C	2	C	2	C	2	C	2
Q21	D or E	if	D	1	D	1	E	1	E	1	E	1
Q22	C	arrays	C	1	C	1	C	1	D	2	C	1
Q23	E	basics	E	1	E	1	E	1	E	1	E	1
Q24	E	recursion	E	1	E	1	E	1	E	1	E	1
Q25	B	basics	B	1	B	3	C	7	B	1	B	3
Q26	B	logical operator	B	3	B	1	B	1	B	1	A	2
Q27	D	function parameters	E	2	D	1	D	1	D	1	A	2

To investigate students ability to demonstrate their understanding in the pseudo-code assessment, the analysis will focus specifically on the instances of codes 1, 2, 6, and 7. The first two codes are clear mappings to students ability to reason with the pseudo-code and answer the questions correctly or incorrectly based upon their understanding of the concept. Codes 6 and 7 represent situations where the question itself, either in its wording or the expression of the code in the pseudo-code syntax, may have confused the student participant. Codes 4 and 5 contribute to an evaluation of the validity of the exam, and will be discussed in Chapter 5. Participant responses for all of the questions followed a reasoned, but not necessarily correct, logic about particular constructs, so the last item in the rubric (code 8) was not identified in the data set.

The majority of the participants answers (83.70%) were judged to be category 1 or 2, a score that indicates the participant was reading and reasoning with the pseudo-code as intended. The following excerpt from participant *M7* demonstrates a correct reasoning and understanding of a *while* loop, expressed in pseudo-code, but without the index variable incremented in the body of the loop, thus creating an infinite loop.

> *M7*: Looking at this while loop, the number i, or variable i is never incremented, so i will always be less than 10 so the while loop will run endlessly. And because i is never incremented, and it always wants to print number and then concatenated with i it will always print number one, number one, etc, um so that would be C.

However, participant *M3* assumed that the index variable was incremented, perhaps automatically as in the definition of for loops, and expressed common misconceptions in their[1] reasoning.

[1]The word "they", often called the singular they, will be used to indicate a gender neutral pronoun.

> *M3*: When i is less than 10, print i = number 1 + i. So then it would be number 2 and then number 3, all the way to number 9. This seems correct. ...
>
> i is always equal to 1, which is less than 10. That's incorrect because you add i to it every time. i is always equal to 1, which is less than 10; so print i is a finite [*sic*] number of times. ... That's also incorrect, because well the answer has 10 in it and it's saying it goes an infinite number of times and it will stop after i gets to 9, so it will not continue to go.

There was no noticeable difference between participants of high and low ability on this metric. High ability participants had an average of 22.5 questions rated in the top two categories, and low ability participants had an average of 22.67 questions in these categories. While the high ability participants did score better on the assessment, and thus were more likely to receive 1's for answering the question correctly, there was no measurable difference in the participants ability to reason with the pseudo-code.

The remaining answers were divided among the rest of the rubric scoring codes. However, there is another small cluster of scores (code 3, 5.92%) that appeared as an artifact of the testing format. In these questions, participants expressed clear misconceptions or incorrect logic about a concept, but the multiple-choice question format of the exam, allowed them to guess and select the correct answer.

In a few instances (n = 7, 5.18%), the wording of the question contributed to participant error. For three questions, the English phrases that were used to describe program behavior led to confusion. One student was confused by the specific term "increment" applied to the behavior of changing the index variable in a loop (Q1), correctly noting that the index variable does not have to be incremented in all for loops. Another student, when reasoning about execution of an if statement (Q6), inferred the phrase "always print" to imply a loop, rather than the intended meaning of being outside the body of the if statement.

P1: If I understand this right, I think A is saying always print "Going on!" like infinitely many times, which its not in the loop so I don't know why it would do that.

When evaluating a recursive function call (Q14), one student was not clear whether the term "final result" implied the return value after the first call to the function or the value after the recursion was complete.

The wording of the stem in question 12 was problematic in multiple ways. Some participants did not notice the use of the phrase "always true" when evaluating the answer choices, and the term "declared variable" was not language independent and especially problematic for Java participants.

The student participants had little difficulty reading and reasoning with the pseudo-code syntax. Two areas in the pseudo-code definition suffered from ambiguity, but there was only one instance where the syntax of the pseudo-code was the source of the participant's misunderstanding. Students were occasionally confused about the declaration and initialization of variables. Java students seemed particularly troubled by the lack of explicit type declarations, and the method for declaring an array was inadvertently omitted from the pseudo-code guide. Participant *P1* talks here about their uncertainty about arrays.

P1: So A is going to turn into - I think, I can't really remember. I think it's like four zeros in a list.

However, as this excerpt demonstrates, except in one case, participants uniformly made correct assumptions about the pseudo-code behavior and proceeded to reason about the item construct.

Question 25 was the one instance where the pseudo-code syntax seemed to distract the student. Q25 asked students to write a math equation in programming syntax. Participant *J4* struggled with how to express "π" and debated what other programming languages allowed.

> *J4*: Well you don't have pi – pi is not just some magic number that we have here so I'm going to assume that we have to actually use 3.14 'cause you can't just put the letter or the Greek letter pi in. I don't think. ...
>
> I'm not sure if I can actually use just pi 'cause I know in Java we can't just put in the symbol pi. ... So maybe the pi thing was right. ... I don't know if you can do that 'cause it's so inaccurate to use 3.14, but I don't know if I can use pi either. ...
>
> I really don't like this 'cause we can never use pi in Java and I'm trying to remember for the little bit of stuff I've done in Python if – I don't know if you can use pi. I don't know if I ever did anything with math for that. ... So I'll leave that one as C even though I don't like using pi.

The think aloud interview data does confirm that students are able to express their understanding of introductory computing concepts in pseudo-code. However, the claim that the majority of errors made would be conceptual, rather than syntactic in nature, is unresolved. The pseudo-code has been designed to mitigate against a difficult syntax learning curve. However, data gathered in this study was insufficient to investigate this hypothesis, and it is unclear whether it would be possible to collect reliable measures of this phenomenon. Computing concepts are expressed using programming language notation, and syntax inherently embodies conceptual constructs. Concepts and syntax are integral components of programming knowledge, therefore it would be difficult to find evidence that would directly link the error to either a specific conceptual or syntactical mistake.

4.3.4 Study 4 Findings and Contributions

- Design of a pseudo-code that is accessible to novice students from a variety of programming language backgrounds.

- Evidence that students are able to read and reason about FCS1 Assessment concepts in the pseudo-code language. (H5, confirmed)

4.4 FCS1 Assessment Study

The final test of programming language independence is whether students were able to demonstrate comparable levels of conceptual understanding in pseudo-code and in their "native" programming language. The research question being investigated was as follows:

RQ2.3: *To what extent are students able to transfer their understanding of fundamental CS1 concepts from their introductory programming language of instruction to pseudo-code?*

H7: Students' ability to transfer their conceptual understanding from their introductory programming language of instruction to pseudo-code will be demonstrated by a positive correlation between scores on the pseudo-code and language versions of the FCS1 assessment.

H8: An aptitude-treatment interaction will influence the degree of transfer exhibited, with higher ability students demonstrating a higher degree of transfer.

I conducted a large scale empirical study comparing student performance on the FCS1 Assessment to a comparable version of the assessment instrument written in the students' CS1 programming language. Participants were selected at the end of CS1 courses taught in Java, Matlab, or Python. Statistical analysis techniques enabled me to look for correlations in student performance between the two exams.

4.4.1 Participants and Recruitment

Participants were recruited as they were completing CS1 courses taught in Java, Matlab, and Python. In particular, I recruited participants from four different introductory courses taught at two universities by four separate faculty members, so

Table 11: Summary of Participants in FCS1 Assessment Study

CS1 Programming Language	University	CS1 Approach	n
Java	C1	Traditional	80
Matlab	C2	Engineering	250
	C2	Engineering	270
Python	C2	CS-Based	71
	C2	Media	139
	C2	Media	142

that the definition and understanding of CS1 knowledge was not tied to a particular faculty member or institution. (See Table 11 for a summary of participant recruitment.) There were a total of 952 participants who were enrolled in a CS1 course in Java ($n = 80$), Matlab ($n = 520$), or Python ($n = 352$).

4.4.2 Study Method

The study consisted of two assessment exams given under testing conditions – no questions were permitted and collaboration was not allowed. Participants completed the FCS1 Assessment and a comparable version of the FCS1 Assessment rewritten in the programming language used in their introductory CS course. The comparable version was created using the alternate questions for each concept in the test bank.

A counterbalanced quasi-experimental design was used to reduce bias from ordering effect. Random assignment of participants to treatment groups is not feasible in this setting, so class sections and test administrations were assigned to the experimental conditions. The FCS1 Assessment was administered as a regularly scheduled course activity during normally scheduled course times (i.e., lecture or recitation section) for the Java and Python participant groups, with students electing to have their data considered for inclusion in the study. Students in the Matlab participant group were also invited to participate in the study, but due to scheduling constraints, the exam was administered outside of normal class meeting times. All students earned extra credit in their CS1 course for their participation.

Approximately half of the students in each programming language participant

Table 12: FCS1 Assessment Final Data Set by Counterbalanced Participant Groups

CS1 Programming Language	**First Exam**	**Pseudo-code** n	**Language** n
Java	Pseudo-code	41	37
	Java	38	37
Matlab	Pseudo-code	295	277
	Matlab	219	214
Python	Pseudo-code	202	156
	Python	136	129

group received the pseudo-code language assessment first. After an interval of 1 week, they received the CS1 programming language version of the exam. The other half of students completed the assessments in reverse order, receiving the CS1 programming language exam first. (See Table 12.)

4.4.3 Data Analysis

Two kinds of statistical data analysis techniques were used to evaluate the claims of this study. Correlation studies were used to investigate whether students demontrated comparable levels of conceptual knowledge in the pseudo-code FCS1 Assessment and in the CS1 programming language versions of the exam. At the end of a first course in computing, the student participants should have learned enough abstract conceptual information to be able to transfer that knowledge into the new pseudo-code syntax. In addition, analysis of variance techniques were used to investigate whether students' ability in CS1 is a predictor of their success on the the FCS1 Assessment.

4.4.3.1 Preparing Data for Analysis

Before data analysis could begin, outliers from the data set that would bias or skew the results were removed. The first set of anomalous data came from participants who did not take the assessment seriously and were only participating to earn the extra credit points. An objective set of rules for exclusion was developed and applied to the data. The following conditions were evidence of an incomplete exam that was thus removed from the data set.

- Participant finished quiz in in less than 15 minutes, allowing approximately 30 seconds or less per question.
- Participant filled in the answer sheet following a clear pattern visually on the page, commonly referred to as "christmas tree-ing" the exam.
- Participant provided the same answer, including blank, to 10 or more questions in a row.
- Participant left 15 or more of the exam questions blank, over half of the exam.

A total of 21 exams were excluded under these conditions, 17 of which were the pseudo-code version of the exam. An external researcher verified the rules for exclusion and independently reviewed all of the exams that were removed from the data set to confirm that they met one or more of the exclusionary criteria. A total of 95 participants only completed the CS1 language version of the FCS1 Assessment, and were also removed from the data set. While data about the pseudo-code version of the exam can be useful to analyze without a corresponding language exam, the language exam score and information is only relevant in comparative analysis to the pseudo-code version. After this initial pass to sanitize the data, there were a total of 931 participants who completed the pseudo-code version of the exam and 850 of those also completed the CS1 language version. Table 12 describes the final participant counts, by programming language participant group.

4.4.3.2 Overview of FCS1 Assessment Scoring Data

The pseudo-code and CS1 language versions of the FCS1 Assessment were graded, awarding a 1 for a correct answer and a 0 for an incorrect answer. (Any question left blank was not scored.) During grading, an error was discovered in question 13 in the pseudo-code exam. The pseudo-code syntax used in the question stem and distractors was not correctly formed. Responses to this item were discarded,

the question was corrected, and the remaining participants were given the modified version of the question. Similarly three questions (Q19, Q25, and Q26) on the first administration of the Matlab version of the assessment had syntax errors, largely stemming from attempting to create equivalent versions of the language specific exam in three different programming languages. Again the responses were discarded, the errors were remedied, and the second half of the Matlab participants received correctly formatted questions.

Table 13: Summary of Graded Pseudo-code Version of FCS1 Assessment

Question	Concept	*Total* # Correct	*Total* % Correct	*Java* # Correct	*Java* % Correct	*Matlab* # Correct	*Matlab* % Correct	*Python* # Correct	*Python* % Correct
Q1	for	265	28.53%	33	41.77%	170	33.20%	62	18.34%
Q2	logical operator	456	50.00%	44	57.14%	268	52.45%	144	44.44%
Q3	while	443	47.63%	49	62.03%	254	49.42%	140	41.54%
Q4	arrays	375	40.85%	45	57.69%	213	41.85%	117	35.35%
Q5	function return values	71	7.63%	14	17.72%	32	6.23%	25	7.40%
Q6	if	637	68.42%	64	81.01%	329	64.01%	244	72.19%
Q7	while	254	27.67%	43	55.84%	109	21.37%	102	30.82%
Q8	for	420	45.45%	49	62.82%	249	48.73%	122	36.42%
Q9	while	331	36.02%	36	46.15%	190	37.48%	105	31.44%
Q10	logical operator	424	46.29%	52	67.53%	261	51.18%	111	33.74%
Q11	function return values	63	6.77%	8	10.13%	18	3.50%	37	10.98%
Q12	basics	18	1.94%	0	0.00%	11	2.14%	7	2.07%
Q13	for	23	29.11%	23	29.11%	—	—	—	—
Q14	recursion	413	44.60%	35	44.87%	281	54.78%	97	28.96%
Q15	function return values	205	22.45%	28	35.90%	118	23.18%	59	18.10%
Q16	function parameters	321	34.59%	24	30.77%	206	40.08%	91	27.08%
Q17	arrays	314	34.93%	43	55.84%	170	33.80%	101	31.66%
Q18	recursion	228	24.92%	17	22.37%	123	24.12%	88	26.75%
Q19	if	322	34.62%	36	45.57%	163	31.71%	123	36.50%
Q20	function parameters	159	17.21%	20	25.97%	81	15.82%	58	17.31%
Q21	if	506	55.42%	50	65.79%	304	59.38%	152	46.77%
Q22	arrays	311	34.59%	40	52.63%	179	35.38%	92	29.02%
Q23	basics	571	62.61%	58	75.32%	342	66.67%	171	53.11%
Q24	recursion	142	15.90%	14	18.67%	100	19.69%	28	9.03%
Q25	basics	492	54.85%	47	62.67%	287	56.72%	158	50.00%
Q26	logical operator	305	34.74%	35	48.61%	174	34.39%	96	32.00%
Q27	function parameters	143	16.00%	15	20.00%	81	16.01%	47	15.02%

— Item not scored due to typographical error in question.

Student participants answered an average of 8.82 (33.78%, $\sigma = 3.649$) questions correctly on the pseudo-code version of the FCS1 Assessment. The maximum score was a 23, and the minimum score was a 1. A summary of the graded assessment, organized by question number, is shown in Table 13. This analysis is not intended to compare results across programming language populations, particularly since the different CS1 courses cover and emphasize different material with respect to the concepts included in the assessment. However, there are a number of questions that appear as the least and most difficult overall and in each programming language population.

Questions focused on introductory concepts were most frequently answered correctly. Questions 23 and 25, that asked students to trace and write statements using the basics of variables, assignment statements, and mathematical operators, were answered correctly by over half of the participants (62.61% and 54.85%, respectively). Questions 6 and 21 were among the easiest on the exam, answered correctly over 55% of the time. These questions had students tracing and predicting the outcome of a series of nested if statements.

One question (Q12) was clearly the most difficult on the exam, with almost no one answering the question correctly, only 1.94% of the participants did so. The question was intended to explore students fundamental notions of variables and assignment statements. However, as discovered in the think aloud interviews, the wording of the question stem was problematic and confusing for most students. Students also struggled with the questions about returning values from functions (Q5 and Q11) with less than 10% of these questions being answered correctly. Correctly tracing a parameterized function call (Q27, 16.00%) and a recursive function (Q24, 15.90%)) also proved challenging for the students.

Table 14: Summary of Graded Language Specific Version of FCS1 Assessment

Question	Concept	*Total*		*Java*		*Matlab*		*Python*	
		# Correct	% Correct	# Correct	% Correct	# Correct	% Correct	# Correct	% Correct
Q1	while	584	69.36%	44	59.46%	357	73.16%	183	65.36%
Q2	for	754	88.81%	69	94.52%	438	89.21%	247	86.67%
Q3	function return values	84	9.89%	6	8.11%	32	6.52%	46	16.20%
Q4	arrays	375	44.17%	37	50.00%	255	51.93%	83	29.23%
Q5	if	739	87.04%	65	87.84%	454	92.46%	220	77.46%
Q6	function return values	160	18.82%	17	22.97%	69	14.05%	74	25.96%
Q7	logical operator	414	49.82%	58	80.56%	261	53.48%	95	35.06%
Q8	if	688	80.94%	65	87.84%	399	81.26%	224	78.60%
Q9	function parameters	205	24.12%	15	20.27%	148	30.14%	42	14.74%
Q10	recursion	189	22.53%	17	23.29%	122	25.05%	50	17.92%
Q11	logical operator	477	56.58%	55	75.34%	323	65.92%	99	35.36%
Q12	if	375	44.22%	45	60.81%	244	49.90%	86	30.18%
Q13	arrays	290	34.48%	45	60.81%	179	36.68%	66	23.66%
Q14	basics	696	81.98%	64	86.49%	417	85.10%	215	75.44%
Q15	basics	588	69.34%	53	71.62%	350	71.43%	185	65.14%
Q16	recursion	503	59.39%	48	65.75%	331	67.55%	124	43.66%
Q17	basics	300	35.29%	48	64.86%	152	30.96%	100	35.09%
Q18	function return values	522	61.41%	54	72.97%	318	64.77%	150	52.63%
Q19	logical operator	245	39.90%	43	58.90%	131	47.81%	71	26.59%
Q20	for	498	59.07%	39	53.42%	341	69.88%	118	41.84%
Q21	for	384	45.77%	45	61.64%	225	46.11%	114	41.01%
Q22	recursion	310	36.99%	34	46.58%	217	44.29%	59	21.45%
Q23	while	482	57.45%	53	72.60%	302	61.76%	127	45.85%
Q24	function parameters	486	58.13%	41	56.94%	344	70.20%	101	36.86%
Q25	while	179	29.20%	32	44.44%	86	31.62%	61	22.68%
Q26	arrays	420	66.99%	62	84.93%	230	83.03%	128	46.21%
Q27	function parameters	210	25.33%	20	27.78%	140	28.81%	50	18.45%

Student participants were more successful answering questions on the CS1 language specific exam. An average of 13.13 (48.61%, $\sigma = 4.195$) questions were answered correctly, and the minimum and maximum score both increased by two points to 3 and 25, respectively. A summary of the language-specific graded assessment is shown in Table 14. While the overall scores were higher on this version of this exam, similar concepts appear among those identified as least and most difficult across the populations. However, there is less consensus across the programming language populations, likely due to the nature of the syntax of the programming languages and how they enable or hinder expression of particular concepts (Weinberg, 1971).

Questions about math operators and if statements were again among the most commonly answered correctly, with over 80% of the participants answering these questions correctly. Two of the questions were the analogous versions of the questions that had appeared in the list of the least difficult items in the pseudo-code exam. Question 14 (81.98%) asked students to write a mathematical equation in programming language syntax, while question 5 (87.04%) asked students to write a series of nested if statements. Students also found the concept understanding questions about both if statements (Q8) and for loops (Q2) to be among the easiest on the exam.

The programming constructs related to function parameters, function return values, and recursion were the most difficult questions on the language specific version of the FCS1 Assessment. Less than 30% of the students were able to correctly answer questions about the behavior of parameterized functions, Q9 and Q27. Question 27 (25.33%), which asked about the values of the parameters during and after a function call, was analogous to Q27, a difficult question from the pseudo-code version of the exam. Another comparable question, Q3 (9.89%), exploring students understanding of function return statements, also appeared as one of the most difficult types of questions on both versions of the exam. Writing code to correctly complete a recursive function to evaluate a string (Q10) was the final concept to be among the most

Table 15: Average Scores on the FCS1 Assessment by Counterbalanced Groups

Pseudo-code Version			
Order	***n***	**Mean**	***σ***
1st	538	8.65	3.619
2nd	388	8.98	3.597

CS1 Language Version			
Order	***n***	**Mean**	***σ***
1st	375	12.55	3.875
2nd	470	13.52	4.345

difficult across all the programming language populations.

After scoring of the exams was complete, additional analysis was conducted to evaluate the effectiveness of the counterbalancing strategy and to measure the equivalence of the divided populations. On the pseudo-code version of the FCS1 Assessment, students who sat for this exam first answered an average of 8.65 questions correctly, and their counterparts who took this version of the exam second averaged 8.98 questions correct (see Table 15). An independent samples t-test was conducted to compare the mean number of correct questions between the testing conditions and yielded no significant difference. This analysis was repeated for the CS1 language specific exam.

Student participants who sat for the CS1 language specific exam first earned an average score of 12.55. Students who took the language exam second, after having completed the pseudo-code version of the exam, earned an average score of 13.52. There was a significant difference in the mean scores for the language exam; $t(843) = 3.382$, $p = 0.001$.

These results suggest that a small learning effect (<= 1 point) was evident in the data. Students taking both versions of the exam second, having seen analogous questions written in a different programming language syntax, did better than their colleagues who completed that version of the exam first. The size of the learning effect is within acceptable limits, and counterbalancing is an appropriate strategy to mitigate any bias introduced by such an effect (Bradley, 1958).

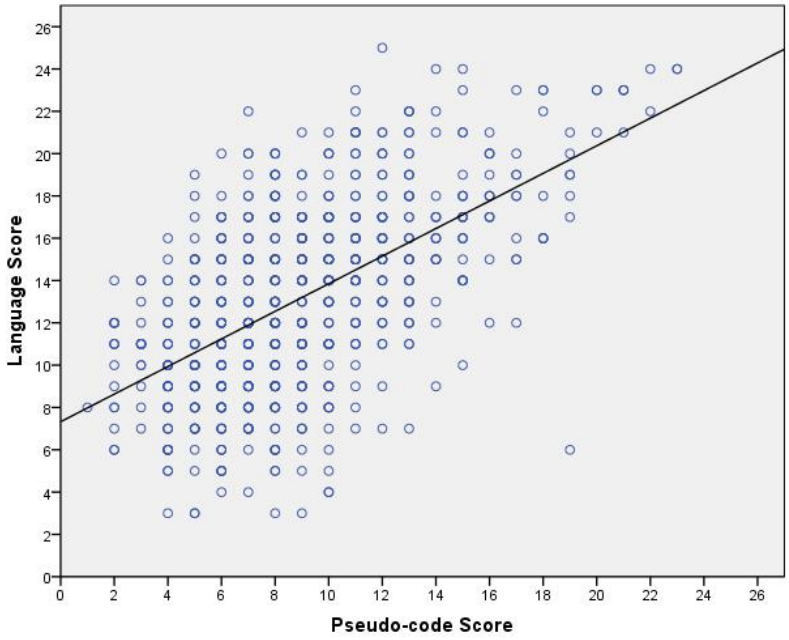


Figure 4: Scatterplot of Scores for Correlation of Pseudo-code and Language Versions of FCS1 Assessment

4.4.3.3 Correlation with CS1 Programming Language Version

To investigate whether students were able to transfer their understanding of fundamental computing concepts from their CS1 programming language to pseudo-code, a correlation analysis was conducted to look for evidence of a positive correlation between the overall scores, i.e. the number of questions answered correctly, for each participant on both versions on the FCS1 Assessment.

A Pearson product-moment correlation coefficient was computed to assess the relationship between the score on the pseudo-code version and the score on the language specific version of the assessment. There was a positive correlation between the two variables, Pearson's $r(850) = .572$, $p <= 0.001$. A scatterplot summarizes the results (Figure 4). Overall for the total participant population, there was a strong, positive correlation between the score on the pseudo-code and CS1 language version of the FCS1 Assessment.

Table 16: Significant Pearson Correlations between Pseudo-code and CS1 Language Versions of the FCS1 Assessment

Population	**df**	***r***	***p***
Total	850	.572	<= 0.001
Java	74	.665	<= 0.001
Matlab	491	.547	<= 0.001
Python	285	.415	<= 0.001
CS-Python	43	.615	<= 0.001
Media-Python	242	.372	<= 0.001

Having found a positive correlation, which meets the guidelines for large effect size in the social sciences (Cohen, 1988), subsequent analyses focused on correlating exam scores for each CS1 programming language population. Were students from each of the CS1 programming languages examined – Java, Matlab, and Python – able to transfer their understanding to pseudo-code? Or is the syntax of the pseudo-code too distinct from what the participants have learned to facilitate the expression of conceptual understanding in a new programming language?

Pearson product-moment correlation coefficients were computed to assess the relationship between the score on the pseudo-code version and the score on the language specific version of the assessment for each programming language participant group (see Table 16). There was a strong, positive correlation between the scores on the pseudo-code and language versions of the assessment for each of the programming languages studied. Participants from the CS1 taught in Java had the strongest correlation, Pearson's $r(74) = .665$, $p <= 0.001$. Although the pseudo-code syntax had more elements in common with Python than the other programming languages studied, the Python participant group had the lowest correlation coefficient, Pearson's $r(285) = .415$, $p <= 0.001$. The Python population was comprised of students, normally students from STEM majors, enrolled in an introductory computing course[2] as

[2]The introductory computing course for CS majors described here covers traditional computer science concepts but is taught in the context of robotics using materials from the Institute for

well as students who were enrolled in a non-traditional media computation course designed for liberal arts students. A Pearson's correlation coefficient was also computed for each of these subpopulations of the Python participant group. There was a strong positive correlation, (Pearson's $r(43) = .615$, $p <= 0.001$), for students enrolled in the CS-based python CS1 course that was similar to the strength of the correlation found in the Java and Matlab populations. While the effect size for the media-based Python course was smaller, $r = .372$, there was still a strong positive correlation.

Overall the results demonstrate that there was a strong, positive correlation between the scores on the pseudo-code and CS1 language versions of the FCS1 Assessment for both traditional and non-traditional pedagogical approaches to CS1.

4.4.3.4 Degree of Transfer to Pseudo-code

The last hypothesis under investigation is whether student ability will significantly impact the degree to which they are able to transfer understanding of fundamental CS concepts from an introductory programming language into pseudo-code. Participant scores on the FCS1 Assessment were re-examined looking for evidence of an aptitude-treatment interaction (Snow, 1989) between a students' success in computer science and the level of transfer exhibited in the assessment.

Given the strength of findings from the the correlation studies, student success in CS was operationalized to be the score on the pseudo-code version of the FCS1 Assessment. The participants in each language population were divided into quartiles: Q1 contains all of the scores in the bottom 25%, Q2 contains scores between the 25th and 50th percentile, Q3 contains scores between the 50th and 75th percentile, and Q4 contains scores in the top 25%. A student's ability to transfer was calculated as the difference between the score on the CS1 language version of the exam and the score on the pseudo-code version of the exam.

Personal Robotics in Education (Summet et al., 2009).

Table 17: Average Scores on the FCS1 Assessment by Quartile

Quartile	n	**Mean Δ**	$\boldsymbol{\sigma}$
4	203	2.31	3.649
3	231	3.76	3.313
2	226	4.81	3.495
1	190	6.20	3.671

Table 18: Post Hoc Comparisons with Bonferroni Correction

Quartile	**Quartile**	**Mean Difference**	$\boldsymbol{p}$
4	3	−1.456	.000
	2	−2.504	.000
	1	−3.895	.000
3	2	−1.048	.006
	1	−2.438	.000
2	1	−1.390	.000

A one-way between subjects ANOVA was conducted to compare the effect of student ability on the ability to transfer (the difference in scores between the CS1 language and pseudo-code versions of the FCS1 Assessment) across the quartile conditions. There was a significant effect of student ability on the difference in scores at the $p < .05$ level for the four quartiles [$F(3,846) = 46.381$, $p <= 0.001$], see Table 17. Post hoc comparisons using the Bonferroni correction indicate that the mean difference in scores for each of the quartiles was significantly less than the next quartile, Q4 < Q3 < Q2 < Q1 (Table 18). That is students of higher ability have scores on the two versions of the exam that are more similar than students of lower ability.

These results indicate that student ability does have an effect on the degree of transfer, with higher ability students better able to transfer from their CS1 programming language to the pseudo-code syntax.

4.4.4 Study 5 Findings and Contributions

- Empirical evidence that students are able to express their understanding of FCS1 Assessment concepts in a language independent assessment. (H7, confirmed)

- Empirical evidence that students across a range of ability levels are able to transfer understanding of FCS1 Assessment concepts from a language specific programming language to pseudo-code. Further, the transfer is influenced by the students ability level, with high ability students demonstrating a higher degree of transfer. (H8, confirmed)

The research evaluating the feasibility of using pseudo-code to achieve a programming language independent exam for introductory CS1 concepts was presented in this chapter. I designed a pseudo-code as the mechanism to express concepts without being tied to a particular CS1 language and conducted a set of studies to evaluate the effectiveness of the pseudo-code as a language for students to express their understanding of CS1 concepts. In so doing, I have addressed research questions RQ 2, 2.1, 2.2, and 2.3, and have presented evidence confirming H4, H5, H7, and H8. The research questions and investigations used to validate the FCS1 Assessment will be discussed in the next chapter.

CHAPTER V

ESTABLISHING VALIDITY

After the Foundational CS1 Assessment was piloted, the final step in the development process is to establish the validity of the exam. 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" (Lindquist, 1951, 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 measuring student knowledge as intended.

Content validity for the FCS1 Assessment was previously established by expert panel review at the end of Study 1. Construct validity is the focus of the research question and hypotheses in this study.

RQ3: *To what extent does the language independent instrument provide a valid measure of CS1 conceptual knowledge?*

H9: There will be a positive correlation between student exam scores and their score on the FCS1 Assessment.

H10: The FCS1 Assessment will provide a valid measure of introductory CS1 content

for procedurally-based CS1 courses taught in Java, Matlab, or Python.

Given the position of this exam as the first of its kind in the field, the standard correlation methods for establishing validity do not apply (Study 2). Since construct validation is dependent on inferences drawn from a variety of data (Kane, 2006; Miller, Linn, & Gronlund, 2009b), I propose a three-pronged approach to establishing the validity of the assessment instrument. Using a combination of think-aloud interview data (Study 4), statistical analysis of student responses in the FCS1 Assessment Study (Study 5), and correlation with exam scores, empirical evidence will be used to argue that the assessment provides an accurate measure of students' understanding of introductory CS1 topics.

5.1 Study Method

The validity study, using the participants and data collection processes previously discussed in Chapter 4, was comprised of three parts. First, the think-aloud interviews (Study 4) allowed investigation into student reasoning while they are answering the questions on the exam. Second, statistical analysis of responses to individual questions in the FCS1 Assessment Study (Study 5) using both the correlation data and item response theory provides evidence that the items are indeed measuring the desired constructs, rather than something more or less than intended. And lastly, an additional study correlating students' scores on the FCS1 Assessment and students' exam scores in their CS1 course. Students' exam scores were used as a measure of their level of understanding of CS1 content as defined by external faculty teaching the course. By recruiting students from multiple institutions, I was able to mitigate the bias of correlating to a particular definition of the content of CS1.

5.2 Data Analysis

The validity evidence is three-fold: think aloud interview data, student responses to the FCS1 Assessment, and student CS1 exam scores. Each of these data sources was analyzed to construct the validity argument for the exam as a whole.

Qualitative analysis of the think aloud transcripts (Study 4) provides evidence whether students were answering questions based on their knowledge of the conceptual content. The goal of the analysis was to determine whether students were using knowledge about a concept to answer the question. Alternatively, additional information could be required to correctly answer the question or other cues could be enabling correct responses without knowledge of the construct.

The FCS1 Assessment study data and analysis provides a quantitative argument towards construct validity. The Pearson correlation analysis (Study 5) demonstrated that students have a comparable knowledge to that measured in a language specific version of the exam. Item response theory and analysis (Baker, 2001) provides empirical evidence towards the quality of the questions themselves. If the questions are shown to be "good" questions (i.e., of appropriate difficulty and discrimination) and students demonstrate comparable knowledge to a language specific exam, then the argument is made that this is an accurate representation of students' understanding of the topics.

Student exam scores provide the final piece of evidence for construct validity. Pearson's correlation analysis was used to investigate whether student scores on the FCS1 Assessment can be positively correlated with their scores on CS1 exams.

5.2.1 Student Think Aloud Interview Responses

Participant responses to the FCS1 Assessment items in a think aloud interview setting were analyzed for evidence of how students derived their answers. Specifically, I was looking for whether students were reasoning about the intended construct or whether

other concepts or cues were causing them to get the questions correct or incorrect.

Eleven questions (40.74%) were answered using valid reasoning (code 1 or 2) about the intended concept by all of the sampled participants. An additional 12 questions (44.44%) had only one participant answer caused by reasoning about another concept or a miscue from the question itself (codes 3 - 7). (See Table 10 for more details). However, there were four items (Q12, Q15, Q17, and Q25) where multiple participants had difficulty reaching valid conclusions based upon the information provided in the question.

Question 12, which has been previously identified as difficult and problematic for students, was frequently coded with the wording of the question as the source of the reasoning error. Questions 15 and 17 saw errors in logic made by reasoning about concepts outside those of the primary focus of the question (code 5). These errors will be discussed in more detail below. A mixture of correct guessing with incorrect logic (code 3) and difficulty reasoning with the pseudo-code (code 7) were included in participant answers for Question 25.

To evaluate validity claims, the analysis will focus on codes 3, 4 and 5, instances where participants are getting the questions right or wrong for some reason other than their understanding of the concept. Overall there were only 10.37% of the answers that were recorded in one of these categories. Codes 1 and 2, previously discussed in Section 4.3, represent valid measures of student understanding. In these instances, student participants are reasoning with the pseudo-code about the intended construct to arrive at a correct or incorrect response. The remaining rubric scores (code 6 and 7) represent issues with the questions or syntax that lead to student error or confusion. These situations confound the expression of understanding and whether a question is accurately measuring the participant's knowledge is inconclusive.

The results of the rubric scoring show eight (5.92%) instances where participants held some misconception or reasoned incorrectly about a concept, yet due to MCQ

format, were able to correctly guess the answer (code 3). This data represents cases where despite not fully understanding a concept, a participant was able to record a correct answer. Therefore these were not valid measures of their understanding of that construct. In most cases (88.89%), the other items about that construct were valid measures, so interpreting across all the questions about a topic should lead to a better understanding of student knowledge.

A common error made by participants appeared in Question 25. When asked to express a mathematical formula in pseudo-code syntax, participants did not notice that distractor D was incorrect because it was missing a multiplication operator. It expressed the formula as commonly seen in mathematics, e.g. , $2(x + y)$ rather than including the operator explicitly required in programming language notation, $2 * (x + y)$. Participant *M7* demonstrates this error and then goes on to explain selecting their answer based on perceived elegance of the solutions presented.

> *M7*: Now what is the difference between B and D? B, two is included inside the parenthesis and D two is excluded. It's outside. Hmmm. Well, if you just follow order of operations, then it all actually works out to be the same two pi r squared for both of them. ... Whereas D, I might have missed something. But still they both seem to work. And I might be just mistaken on this order of operations thing. But I'll pick B. It's simpler and more likely to be correct.

Logical operators were the source of repeated confusion. Individual students struggled on all three boolean logic questions, Q2, Q10, and Q26. Two participants struggled to evaluate basic logical expressions (e.g. $True\ AND\ False\ AND\ True$), expressing little understanding of basic truth tables. The other participant who struggled with this concept demonstrated inconsistent reasoning when working with boolean operators.

On other questions, Q8 and Q16, the student being interviewed made errors but did not recognize the error or know how to correct it. However, the testing format allows them to just guess. On question 8, which focused on tracing nested FOR loops, participant *M3* incorrectly computed the array index from the loop index variables i and j to be -1. They recognized that this value could not be correct, did not know how to resolve the tracing error, so resorted to guessing instead.

> *M3*: All right, so this one writes in reverse and this one does not. ... this j loop has to carry it out to the sequence before it goes back into the i loop, which makes me think it'll write out the word, but what order it writes it out in, I do not know. So I'd have to guess between these two.
>
> I'm going to guess E, because it seems logical they would put it back in the right order. I got -1's, so that makes me think it's going to flip the order of the array of name or something like that.

The final error in this category is made by participant *J4*, who cannot decide what the correct statement is for the base case in a recursive function. After waffling back and forth, they ultimate selecting the correct answer while admitting that it was a complete guess.

Interview codes 4 and 5 were used to represent cases where the correct or incorrect response was reached, not due to a participant's reasoning about the item construct. Rather, reasoning about some other topic led to their response. There were no instances in the data set where a correct answer was caused by reasoning about concepts other than the focus of the question (code 4), and there were only 6 (4.44%) instances where reasoning about a concept other than the primary focus caused a participant to select an incorrect answer choice.

Question 15 was intended to measure participants knowledge about return values.

Stub code was prepared that parsed the input string into an acronym, and the examinees were asked to fill in missing code to return the appropriate values depending on the input string. Two participants were confused about how to determine if the string only contained one word, the conditional clause in the IF statement. Their inability to successfully navigate this problem, confusing the length of the string and the number of words in the string, led to the error.

The modulus operator, %, was the source of difficulty for two participants on question 17. The question focused on how to assign values in arrays of odd or even integers, but difficulty in understanding this mathematical operator prevented students from correctly reasoning about arrays. This excerpt from participant *P1* expresses confusion about the output and evaluation of the operator.

> *P1*: If $numList[i]$, which i equals zero, divided by two equals one, then it's odd. ... Okay, if $numList[i]$ divided by two is greater than zero, it doesn't address anything. If $numList[i]$ divided by two is not zero, which means it's one, then it's going to be odd, which is also even. ...
>
> So. Hmmm. I'm not really sure how to do this one. ... I feel like these [C & E] are both the same and like as in they're just like opposites. They both can't be right. So I guess A is the right answer.

The majority of questions, an average of 22.6 out of 27, were answered using sound logic and reasoning about the intended construct and thus were valid measures of participant understanding. The results indicated a small percentage of false positives expected in a multiple-choice format, and a small number of cases (10.37%) where something other than reasoning about the intended construct led to an incorrect response. Overall, the data suggests that items are generally measuring knowledge of the construct, not something more or less than intended.

5.2.2 Item Response Theory

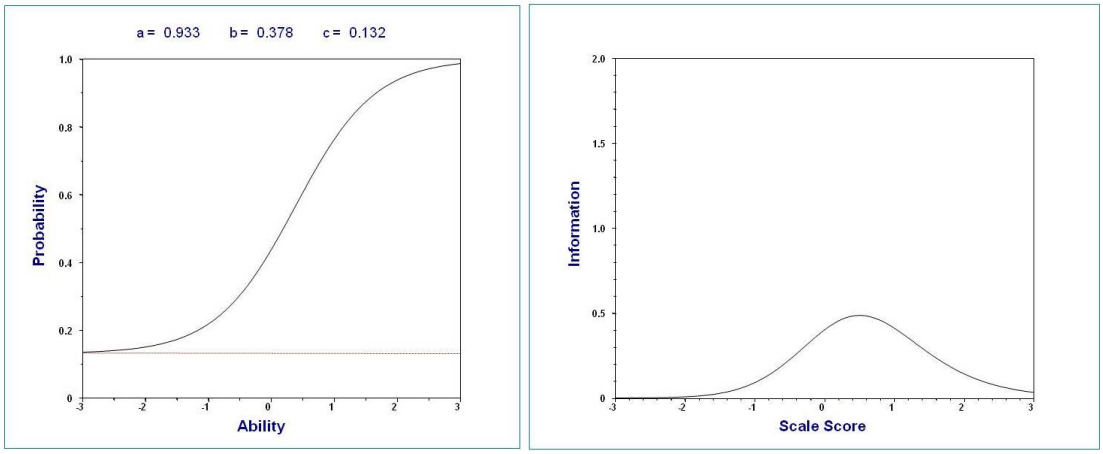
Having found strong positive correlations between the pseudo-code and language specific versions of the FCS1 Assessment, I investigated the quality of the questions themselves. If the questions are representative measures of knowledge in CS1 programming language, then validity of the exam is contingent upon the questions themselves. Are the questions of appropriate difficulty and can they adequately distinguish between students of varying ability levels? Item response theory is the statistical analysis technique employed to make this validity claim.

Item response theory (IRT), by focusing on the question as the unit of analysis, discerns the strength and weakness of each item in a test. The item characteristic function or item characteristic curve (ICC) gives the probability that a person with a given ability, Θ, will answer a question correctly. Using a three-parameter logistic model (3PL) (Hambleton, Swaminathan, & Rogers, 1991), the probability of a correct response to an item i is expressed as:

$$P_i(\Theta) = c_i + (1 - c_i)\frac{e^{Da_i(\Theta - b_i)}}{1 + e^{Da_i(\Theta - b_i)}}$$

$P_i(\Theta)$ is the probability that a participant answers question i correctly. Examinees with higher ability are more likely to get the question correct, and as ability level increases the probability of answering the question correctly increases. The item parameters (a_i, b_i, c_i) are computed using the function and determine the shape of the item curve.

Parameter b_i is referred to as the item difficulty parameter. It represents the point where the probability of a correct response is 50% and the item response function has it maximum slope. In the example ICC shown in Figure 5a, $b_i = 0.378$ which indicates the item is of medium difficulty. The parameter a_i represents the discriminating power of an item — the higher the value the greater the capacity to discriminate between participants at different ability levels. The a_i parameter is characterized by the



(a) Item characteristic curve (b) Item information curve

Figure 5: Item Characteristic and Information Curves for Question 3

maximum slope of the curve, which is at point b_i. The example curve shows an item with a high level of discrimination, $a_i = 0.933$. The curve is steep, with substantial changes in probability of a correct response just a short distance to the left and right of the middle of the curve.

The c_i parameter, the pseudo-chance level or guessing parameter, calculates the probability that a low-ability participant answers the item correctly. In multiple-choice assessments it is used to measure the effect of guessing on the probability of a correct response. In a five-option MCQ, there is a 20% chance of a correct guess randomly, so c_i would be approximately 0.20. In the example shown in Figure 5a, $c_i = 0.132$ and is indicated by the horizontal line at that position. D is a constant equal to 1.7 (Birnbaum, 1968).

In addition, the item information function, $I_i(\Theta)$, plays an important role in test development by computing how much each item contributes to distinguishing an examinees ability. The function (Birnbaum, 1968) is expressed as:

$$I_i(\Theta) = \frac{[P_i'(\Theta)]^2}{[P_i(\Theta)][1 - P_i(\Theta)]} = \frac{2.89a_i^2(1 - c_i)}{[c_i + e^{Da_i(\Theta - b_i)}][1 + e^{-Da_i(\Theta - b_i)}]^2}$$

The item parameters (a_i, c_i, c_i) play a role in determining the information provided by item i. An example item information curve (IIC) is shown in Figure 5b.

Information is higher when the a_i parameter is higher, therefore the contribution depends greatly on an item's discriminating power. Information also increases as guessing (c_i) decreases and when b_i is close to Θ, that is difficulty approaches ability. The place where an item provides maximum information is denoted Θ_{max}.

Taken together, item parameters a_i, b_i, and c_i and item information, $I_i(\Theta)$, provide a basis for distinguishing good and bad items on an assessment instrument. Item response theory, using the 3PL model, was used to analyze the participant responses to the questions in the FCS1 Assessment.

Overall, most (24 of 27) questions displayed strong item discrimination, adequate difficulty, and low guessing probability (see Table 19). Therefore, they showed ideal shapes in their item characteristic and item information curves, see Figure 5. Three questions (Q4, Q6, and Q25) had relatively low item information values, and thus were not making significant contributions to the determination of the student's ability level. Since c_i was generally low, the point of maximum information, Θ_{max}, for all questions was less than 0.50 away from the item's difficulty level, b_i. However, due to the overall difficulty level of the exam, the point of maximum information for 40.74% of the questions was at a high ability level, Θ. That is, the exam questions provided more information about the ability level of high ability students than those of lower ability levels.

Two questions, Q12 and Q13, were too difficult. That is $b_i > 3$, which implies that less than 10% of the examinees had a 50% probability of answering the question correctly, see Figure 6. The difficulty item parameter b_i was 3.979 and 3.473 for Q12 and Q13 respectively. Issues with the wording of question 12 have been discussed previously in Section 4.3 and question 13 was only answered by participants in the Java programming language participant group[1]. It is possible that a larger sample

[1] An error in typesetting Q13 for the Matlab and Python participant groups removed this question from the data set for those populations.

Table 19: Estimated Item Parameters and Information on FCS1 Assessment

Item	Discrimination (a_i)	Difficulty (b_i)	Guessing (c_i)	Maximum Information (I)	Point of maximum I
Q01	0.892	1.234	0.101	0.473	1.339
	0.159	0.122	0.033	0.1401	0.1238
Q02	1.554	1.598	0.442*	0.723	1.767
	0.649	0.167	0.026	0.4337	0.1929
Q03	0.933	0.378	0.132	0.488	0.501
	0.155	0.135	0.053	0.1237	0.1265
Q04	0.491	2.272	0.295	0.099+	2.690
	0.157	0.443	0.054	0.0453	0.4939
Q05	1.564	2.370	0.055	1.587	2.406
	0.523	0.200	0.010	1.0344	0.2090
Q06	0.436	-0.468	0.251	0.0848+	-0.045
	0.082	0.404	0.099	0.0263	0.3658
Q07	1.427	2.180	0.245	0.920	2.307
	0.575	0.219	0.019	0.6351	0.2511
Q08	0.546	1.415	0.274	0.127	1.773
	0.147	0.311	0.067	0.0457	0.3037
Q09	0.823	1.462	0.221	0.320	1.667
	0.218	0.179	0.044	0.1286	0.1876
Q10	0.800	0.865	0.237	0.293	1.087
	0.180	0.195	0.060	0.0888	0.1773
Q11	1.603	2.472	0.052	1.678	2.505
	0.605	0.220	0.009	1.2396	0.2288
Q12	1.130	3.979‡	0.021	0.886	3.999
	0.587	1.082	0.005	0.9167	1.0912
Q13	0.938	3.473‡	0.256	0.389	3.672
	0.450	1.210	0.052	0.3240	1.2500
Q14	0.789	0.559	0.130	0.350	0.704
	0.134	0.151	0.054	0.0906	0.1438
Q15	1.258	1.450	0.108	0.928	1.528
	0.260	0.105	0.023	0.3400	0.1095
Q16	0.687	1.172	0.127	0.267	1.335
	0.135	0.169	0.047	0.0814	0.1687
Q17	0.866	2.255	0.295	0.307	2.492
	0.299	0.303	0.029	0.1671	0.3499
Q18	0.969	2.938	0.229	0.437	3.117
	0.401	0.494	0.019	0.3141	0.5477
Q19	0.802	1.544	0.213	0.309	1.749
	0.199	0.184	0.042	0.1164	0.1900
Q20	1.230	2.315	0.139	0.837	2.412
	0.411	0.226	0.017	0.5131	0.2475
Q21	0.927	0.560	0.309	0.342	0.787
	0.216	0.201	0.065	0.0999	0.1759
Q22	0.834	2.003	0.269	0.300	2.233
	0.254	0.237	0.033	0.1425	0.2685
Q23	0.700	-0.210	0.173	0.254	-0.007
	0.107	0.206	0.075	0.0630	0.1912
Q24	1.142	1.691	0.065	0.830	1.748
	0.206	0.113	0.017	0.2753	0.1170
Q25	0.496	1.465	0.390	0.083+	1.958
	0.150	0.424	0.070	0.0311	0.4166
Q26	1.011	1.507	0.237	0.468	1.682
	0.232	0.157	0.034	0.1672	0.1607
Q27	1.594	2.297	0.138	1.407	2.372
	0.643	0.214	0.014	1.0589	0.2342

‡ Item exceeds recommended difficulty.

* Item exceeds recommended guessing probability.

+ Item fails to provide adequate information.

Table 20: Estimated Item Parameter Means and Standard Deviations

Parameter	Mean	Standard Deviation
a_i	0.975	0.348
b_i	1.746	1.018
c_i	0.195	0.102

is needed to fully evaluate the difficulty of this item, but the analysis indicated that both questions need to be revised before being included in future versions of the assessment.

Overall, the items on the exam showed adequate levels of discrimination. No values of the a_i item parameter were 2 or more standard deviations from the mean, ($a_i = 0.975$, $\sigma = 0.348$). (See table 20). Four questions (Q4, Q6, Q8, and Q25) were greater than 1 standard deviation from the mean and thus could possibly be improved to be better discriminators. In general, due to the overall high level of difficulty of the exam, the test shows better discrimination among those of higher ability ($\Theta > 1$) than those with lower ability ($\Theta < -1$).

Question 2 displayed a guessing probability that exceeded recommended limits. The guessing item parameter, $c_i = 0.442$, a probability of a low ability participant guessing the question correctly over 40% of the time. (See Figure 7). Two other questions, Q21 and Q25, had guessing parameter values that were elevated but were within one standard deviation of the expected value of c_i =0.20 for a 5 item multiple-choice question.

Three questions (Q4, Q6, Q25) had low item information values, less than 0.25, and therefore contributed little to the overall ability level of the student participant. The item information curve for question 6 is included in Figure 8b. (Item characteristic and information curves for the remaining questions are included in Appendix D.)

Overall, IRT analysis identified four questions, Q2, Q12, Q13, and, Q25 that need to be revised or dropped from the exam. Question 2 has a high guessing probability,

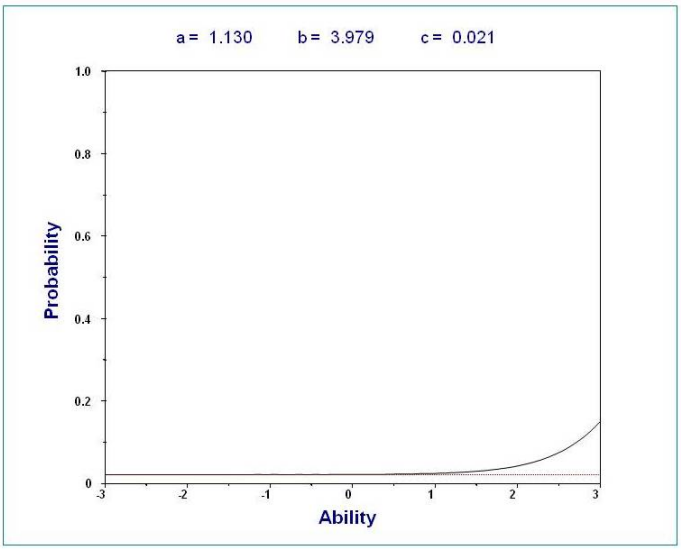


Figure 6: Item Characteristic Curve for Question 12

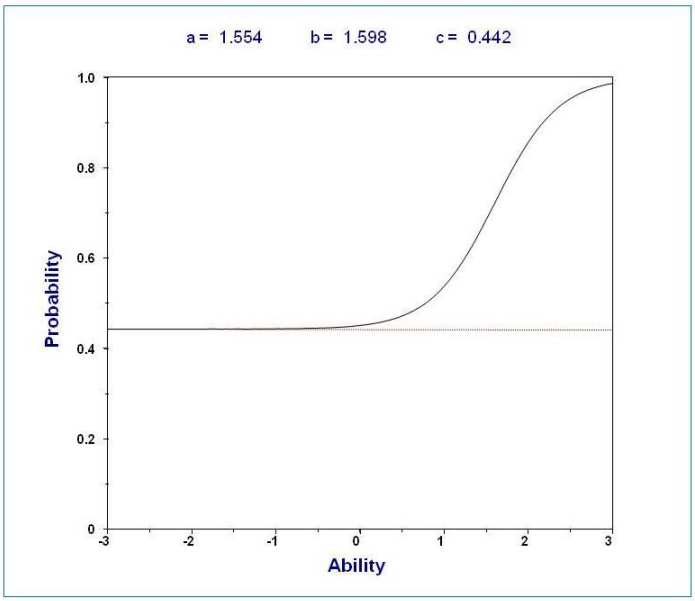
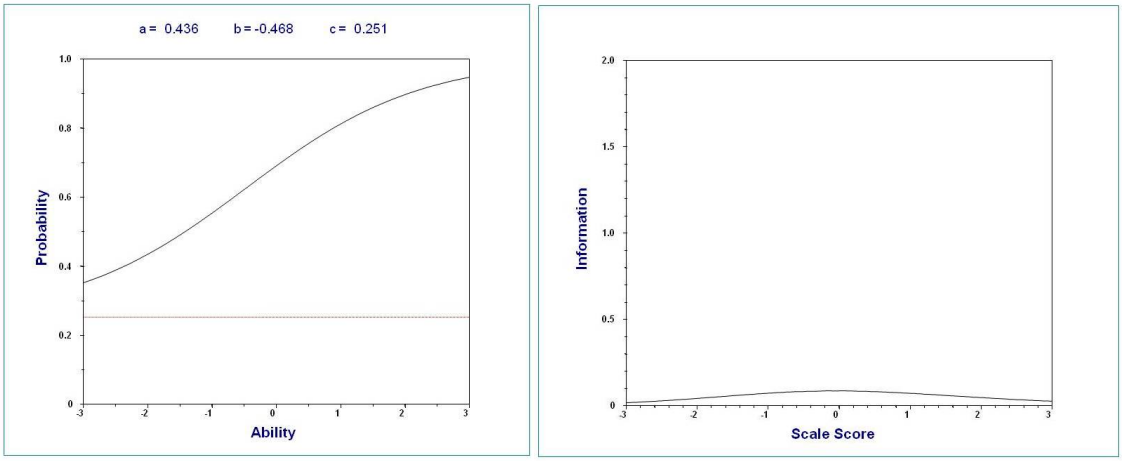


Figure 7: Item Characteristic Curve for Question 2



(a) Item characteristic curve (b) Item information curve

Figure 8: Item Characteristic and Information curves for Question 6

questions 12 and 13 are too difficult, and question 25 has a relatively high guessing probability and provides low information. In addition questions 4 and 6 are candidates for improving discriminating ability, which will subsequently improve item information.

A natural source for question revision is the alternate versions of the questions that are included in the test bank that have already undergone formative testing, in the language specific format in the FCS1 Assessment study (Study 5). Questions 6, 13, and 25 can be replaced with the questions from the test bank that piloted at better difficulty and discrimination levels in each of the programming language populations. Question 12, due to its high level of difficulty and problems identified in the question wording, can also be replaced with the alternate question from the test bank. Although the alternate may need further refinement based upon its initial pilot testing. The current versions of questions 2 and 4 will need to be revised as the test bank questions fared worse in pilot testing.

5.2.3 Correlation with Student Exam Scores

To investigate whether the FCS1 Assessment was measuring student understanding similarly to existing definitions and measures used in CS1 courses, I conducted a correlation analysis. The analysis looked for evidence of a positive correlation between the overall score on the assessment and participant exam grades in their introductory course.

A Pearson product-moment correlation coefficient was computed to assess the relationship between the score on the FCS1 Assessment and the score on the final exam in CS1. There was a positive correlation between the two variables, Pearson's $r(931) = .499$, $p <= 0.001$. A scatterplot summarizes the results (Figure 9). Overall for the total participant population, there was a strong, positive correlation between the score on the pseudo-code and CS1 language version of the FCS1 Assessment. Further there were significant, yet weaker, correlations between scores on the assessment and scores on individual midterm exams (see Table 21). Exam scores on midterm 1 and 2 were weakly correlated ($r <= .150$), while mid-term 3 showed evidence of a moderately strong correlation (Pearson's $r(931) = .309$, $p <= 0.001$).

Having found a strong positive correlation, subsequent analyses focused on correlating exam scores with each CS1 programming language population. Pearson product-moment correlation coefficients were computed to assess the relationship between the score on the pseudo-code version of the assessment and the final exam score for each programming language participant group (see Table 21). There was a strong, positive correlation between the scores on the assessment and the final exam for each of the programming languages studied. Participants from the CS1 taught in Java had the strongest correlation with final exam score, Pearson's $r(79) = .511$, $p <= 0.001$. The Python population was again comprised of students in CS and media computation versions of CS1. A Pearson's correlation coefficient was computed for each of these subpopulations of the Python participant group. There was the strongest

Table 21: Significant Pearson Correlations of FCS1 Assessment‡

Population	Version	Language	Midterm 1	Midterm 2	Midterm 3	Final
Total	Pseudo	.572	.111	.149	.309	.499
	Language	*	.120	.128	.491	.542
Java	Pseudo	.665	.408	.446	—	.511
	Language	*	.406	.582	—	.680
Matlab	Pseudo	.547	.500	.527	.443	.438
	Language	*	.531	.536	.488	.505
Python	Pseudo	.415	.200	.216	*n.s.*	.453
	Language	*	.424	.429	.392	.539
CS-Python	Pseudo	.615	.719	.685	.625	.679
	Language	*	.525	.445†	.429†	.437†
Media-Python	Pseudo	.372	.246	.220	.305	.262
	Language	*	.456	.453	.542	.601

‡ All correlations, unless otherwise noted, are significant at the $p <= 0.001$ level.
† Correlation is significant at the $p <= 0.01$ level.
* Language-Language self comparison omitted.
— No midterm 3 measure was analyzed because only two midterms were administered in the course.

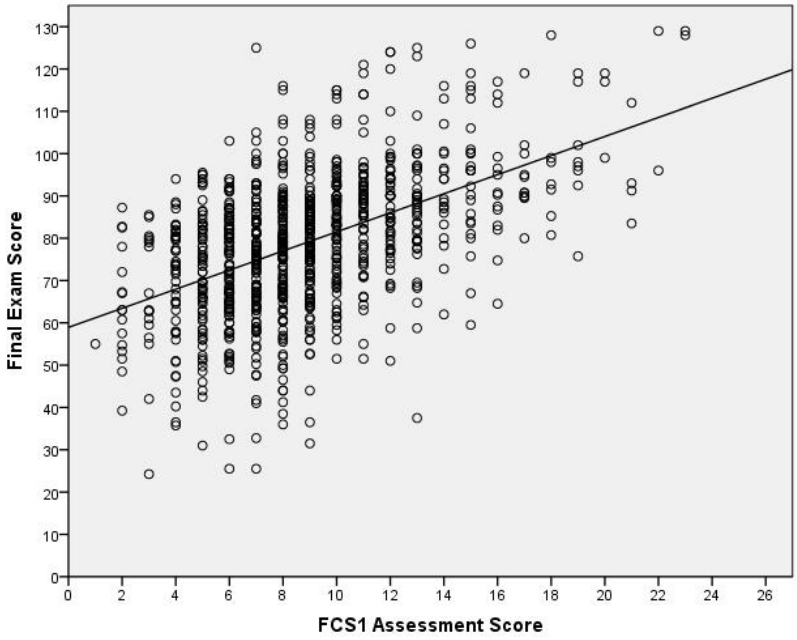


Figure 9: Scatterplot of Scores for Correlation of FCS1 Assessment Score and CS1 Final Exam Score

positive correlation, (Pearson's $r(69) = .679$, $p <= 0.001$), for students enrolled in the CS-based python CS1 course. The effect size for the media-based Python course was smaller, $r = .262$, yet there was still a significant positive correlation.

Pearson correlation coefficients were also computed for each midterm score for each programming language participant group (see Table 21). There was a strong, positive correlation between the scores on the assessment and midterm exam scores for all CS1 courses in Java, Matlab, or Python except the non-traditional media computation approach. Participants from the CS-based Python course had the strongest correlations with each midterm exam score. The strongest correlation coefficient was for midterm 1, Pearson's $r(69) = .719$, $p <= 0.001$. The correlation between assessment score and midterm exam scores for the media-based Python course showed significant correlations with medium effect size. For comparison, the correlation coefficient for midterm 1 in the media Python population was Pearson's $r(269) = .246$, $p <= 0.001$.

Overall the results demonstrate that there was a strong, positive correlation between the scores on the FCS1 Assessment and final exam scores for both traditional and non-traditional pedagogical approaches to CS1. In addition there was a strong positive correlation with individual midterm exam scores for traditional approaches to CS1 taught in Java, Matlab, and Python.

5.2.4 Validity Argument

Two issues are central to construct validation of an assessment instrument: *construct under-representation* and *construct-irrelevant variance* (Miller et al., 2009b). These issues are explored by the following questions: (1) Does the assessment adequately operationalize the intended construct? and (2) Is performance on the assessment influenced by factors that are ancillary to the construct?

The matter of construct under-representation was resolved by the panel of expert reviewers confirming an adequate definition of fundamental CS1 concepts included in the test specification. Further, item response theory analysis indicated that a majority (24 out of 27) of the items on the assessment provide adequate information about student participant ability. Thus, overall the definition and measurement of the constructs specified are appropriate for the FCS1 Assessment.

A variety of metrics were used to identify potential sources of construct-irrelevant variance, with most measures providing evidence to the contrary. Think aloud interviews with participants revealed that students were able to provide valid answers about the intended construct on over 85% of the questions. Scores on the pseudo-code version of the assessment had a strong positive correlation with scores on the CS1 language specific version of the exam. When combined with IRT results that demonstrate that 85.18% of the questions were of appropriate difficulty and discrimination, it is appropriate to infer that the FCS1 Assessment is a reasonable measure of CS1 knowledge.

Overall the validity studies provide evidence that students are reading and reasoning with the pseudo-code to answer questions in the manner intended. In addition, there is empirical evidence of the quality of the questions used to measure understanding that further correlates with external faculty definitions and measures of CS1 knowledge.

5.3 Study 6 Findings and Contributions

- Empirical evidence of a positive correlation between student CS1 exam scores and scores on the Foundational CS1 Assessment. (H9, confirmed)
- The Foundational CS1 Assessment provides a valid measure of introductory CS1 content for declarative programming languages, Python, Matlab, and Java. (H10, confirmed)

In this chapter, I have proposed a method for constructing the validity argument for the FCS1 Assessment. A combination of evidence was used – think aloud interviews, statistical analysis of student responses, and correlation with exam scores. This study addressed research question RQ3 and presented evidence confirming hypothesis H9 and H10. A summary of findings and contributions and a brief discussion of future work is presented in the next chapter.

CHAPTER VI

CONCLUSION AND FUTURE WORK

The goal of assessment research in computer science is to have valid ways of measuring student conceptions of fundamental topics, which will enable both research into how understanding of knowledge in the domain develops as well as enable curricular innovation and reform grounded in this knowledge. This thesis focused on three research questions regarding assessment of introductory concepts in computer science.

RQ1: *How can existing test development methods be applied and adapted to create a valid assessment instrument for CS1 conceptual knowledge?*

RQ2: *To what extent can pseudo-code be used as the mechanism for achieving programming language independence in an assessment instrument?*

RQ3: *To what extent does the language independent instrument provide a valid measure of CS1 conceptual knowledge?*

As was demonstrated across the six studies, classical test development methods can be adapted and applied to a disciplinary specific field, such as computer science, to create a valid assessment instrument. Modifications to the recommended process may be necessary due to specific domain or notational constraints, but the standard guidelines (American Educational Research Association et al., 1999) provide a solid foundation for starting development on a new assessment instrument. For computer science, two adaptations were necessary. Given the goal of creating an exam that would be as widely applicable as possible, an additional step was needed to verify that pseudo-code was an appropriate mechanism for achieving programming language independence. The method for establishing the validity of the instrument also had

to be supplemented with additional evidence beyond traditional correlation studies, since the exam was the first of its kind in the field.

The think aloud interview and FCS1 Assessment studies demonstrated that pseudo-code was an appropriate mechanism for assessing fundamental CS1 concepts in a programming language independent manner. Students, regardless of the programming language used in their introductory course, expressed similar conceptual errors on open-ended questions, and these errors informed the design of the distractors in the multiple-choice format questions. In think aloud interviews, examinees displayed the ability to read and reason in the pseudo-code syntax without difficulty. In a large-scale empirical study, students were successfully able to transfer conceptual knowledge from their CS1 programming language to pseudo-code.

Finally, validity of the assessment for students enrolled in a procedurally-based introductory computing course taught in Java, Matlab, or Python was established using a multi-faceted argument. Think aloud interviews confirmed that student participants were answering questions based on their knowledge of the conceptual content. Item response theory validated the quality of the questions used in the correlation study, and the FCS1 Assessment scores positively correlated with external faculty definitions and measures of CS1 content.

6.1 Contributions

In answering these research questions, this research makes several contributions to the field of computer science education. First, I have provided an example of how to bootstrap the process for developing the first assessment instrument for a disciplinary specific design-based field. Through the validation studies, I have identified that as in other fields, computer science assessment instruments specify different kinds of knowledge to be examined and for different purposes. Therefore it may not be possible to correlate scores between exams created with different measurement goals, yet that

does not necessarily diminish the validity claims of the individual assessments.

I have also demonstrated that novice computing students, at an appropriate level of development, can transfer their understanding of fundamental concepts to pseudo-code notation. This ability enables assessment across and comparison of pedagogical approaches. Lastly, I have built a valid assessment of introductory computing concepts for procedurally-based introductory computing courses taught in Java, Matlab, or Python at the university level.

6.2 Future Work

Research and development on the Foundational CS1 Assessment can continue along a number of paths. This research has focused on establishing the validity of the exam for a limited number of constructs with a focused population of university students studying common introductory CS1 programming languages. Natural extensions of this work include adding the tenth common topic identified in Study 1, object-oriented basics; establishing the reliability of the exam; and implementing the test on-line to reduce the resources required for data collection and test administration. I would also like to explore the applicability and validity of using the FCS1 Assessment for measuring student knowledge across different pedagogical paradigms and programming languages, such as graphical or functional approaches. For example, will students who learn to program in a graphical drag and drop interface, such as Alice (Dann, Cooper, & Pausch, 2006), be able to transfer their conceptual understanding to the pseudo-code syntax? Hundhausen, Farley, and Brown (2009) found evidence of early transfer from direct manipulation interfaces to textual programming which suggests this approach may be feasible.

The availability of a valid assessment instrument to measure student understanding of CS1 concepts enables a variety of directions for CS education research involving the FCS1 Assessment. Two of particular interest are discussed here.

A programming language independent assessment instrument permits the comparison of pedagogical approaches in ways that were not previously available. In particular, it is now possible to investigate whether there are identifiable and persistent differences in student understanding of fundamental computer science concepts based upon the pedagogical approach or programming language used in the first course. When combined with other research methods, it would be possible to begin to identify which of the many factors in a CS1 learning environment (e.g., the teacher, programming language, integrated development environment (IDE), pedagogical approach, student motivation) are the levers that drive student mastery of computing concepts.

The Foundational CS1 Assessment, as a programming language independent measure of conceptual knowledge, also starts to permit the comparison of programming languages used for novices. Given a carefully constructed study, the assessment could be used to investigate which concepts a particular programming language elucidates for beginning programmers and which concepts are more difficult to learn in a language (Kelleher & Pausch, 2005). The assessment could also be used in the design and evaluation of claims about new programming languages and environments specifically designed for novice programmers.

Assessment is an important piece of the computer science education research agenda. Moss, Girard, and Haniford (2006) remind us of the broader implications:

> we must recognize that assessment practices do far more than provide information, they also shape people's understanding about what is important to learn, what learning is, and who learners are. (p. 111)

Valid measures of computing concepts for pedagogical and research purposes enable inquiry into the nature of learning in the discipline and help us move closer to being able to identify and predict successful models for developing domain expertise.

APPENDIX A

ASSESSMENT INSTRUMENT

Unfortunately for the validity and reliability of the assessment instrument, exam questions must remain private and cannot be published. This prevents potentially biasing participants involved in the validation studies. Paper copies of the questions were made available to committee members, and exemplars were provided at the defense.

APPENDIX B

STUDY 3 DATA - OPEN ENDED FCS1 QUESTIONS STUDY

This appendix contains the data from the analysis of the open-ended versions of the FCS1 Assessment questions presented in Chapter 4.

Table 22: Common Incorrect Answers to Closed-Form Question Q2 Ranked by Frequency of Occurrence

variable: *first*, 75.6% correct						
	Java		**Matlab**		**Python**	
	Value	n	Value	n	Value	n
1st	$0^{\ddagger}$	4	$0^{\ddagger}$	8	$0^{\ddagger}$	3
2nd	2^{*}	2	6	5	$3^{\diamond}$	3
3rd	4^{*}	2	2^{*}	2	4^{*}	3
4th	$18^{\diamond}$	2	$3^{\diamond}$	2	8	2
5th			$18^{\diamond}$	2		

variable: *previous*, 80.2% correct						
	Java		**Matlab**		**Python**	
	Value	n	Value	n	Value	n
1st	2^{*}	4	$0^{\ddagger}$	8	$1^{\ddagger}$	5
2nd	$0^{\ddagger}$	3	$1^{\ddagger}$	3	2^{*}	5
3rd	$1^{\ddagger}$	2	6	2	$0^{\ddagger}$	2
4th					4	2

variable: *final*, 75.6% correct						
	Java		**Matlab**		**Python**	
	Value	n	Value	n	Value	n
1st	$0^{\ddagger}$	6	$0^{\ddagger}$	8	$0^{\ddagger}$	6
2nd	$2^{\ddagger}$	3	$2^{\ddagger}$	4	1	3
3rd	$3^{\ddagger}$	2	$3^{\ddagger}$	3	$2^{\ddagger}$	3
4th					$3^{\ddagger}$	2

‡ Trend identified across all three languages.
* Trend identified across two languages, but answer appeared in all groups.
$^{\diamond}$ Trend identified across two languages.

Table 23: Common Incorrect Answers to Closed-Form Question Q5 Ranked by Frequency of Occurrence

variable: *answer*1, 87.13% correct						
	Java		**Matlab**		**Python**	
	Value	n	Value	n	Value	n
1st	false‡	6	false‡	8	false‡	2
2nd	error	2			true and false	2

variable: *answer*2, 91.76% correct						
	Java		**Matlab**		**Python**	
	Value	n	Value	n	Value	n
1st	true‡	7	true‡	3	true‡	2
2nd	error	2				

variable: *answer*3, 75.6% correct						
	Java		**Matlab**		**Python**	
	Value	n	Value	n	Value	n
1st	true‡	11	true‡	10	true‡	5
2nd	error/neither*	5			error/neither*	4
3rd	true and false*	2			true and false*	3

‡ Trend identified across all three languages.
* Trend identified across two languages, but answer appeared in all groups.
◇ Trend identified across two languages.

Table 24: Common Incorrect Answers to Closed-Form Question Q14 Ranked by Frequency of Occurrence

variable: i, 39.35% correct						
	Java		**Matlab**		**Python**	
	Value	n	Value	n	Value	n
1st	5*	7	9◇	15	1‡	12
2nd	8	7	1‡	13	0◇	4
3rd	0◇	2	5*	6	10◇	3
4th	1‡	2	4	3		
5th	9◇	2	7	3		
6th			10◇	3		

variable: *even*, 44.08% correct						
	Java		**Matlab**		**Python**	
	Value	n	Value	n	Value	n
1st	4◇	8	0‡	11	1*	13
2nd	0‡	5	4◇	9	0‡	7
3rd	5◇	2	5◇	7		
4th			1*	4		
5th			2	4		

‡ Trend identified across all three languages.
* Trend identified across two languages, but answer appeared in all groups.
◇ Trend identified across two languages.

Table 25: Common Incorrect Answers to Closed-Form Question Q17 Ranked by Frequency of Occurrence

variable: x, 5.71% correct						
	Java		**Matlab**		**Python**	
	Value	n	Value	n	Value	n
1st	12‡	49	12‡	59	12‡	18
2nd	0	3	6*	4	14	3
3rd	6*	2	7	2	$x + 2$	3
4th	15	2	s	2	$x + length(y)$	3

variable: y, 5.92% correct						
	Java		**Matlab**		**Python**	
	Value	n	Value	n	Value	n
1st	spiderman‡	47	spiderman‡	54	spiderman‡	23
2nd	9	2	n	3	$y +$ "man"*	5
3rd	man◇	2	0	2	man◇	2
4th	spider	2	$y +$ "man"*	2	$y + 2$	2

variable: s, 72.73% correct						
	Java		**Matlab**		**Python**	
	Value	n	Value	n	Value	n
1st	12‡	13	12‡	20	12‡	6

variable: n, 60.40% correct						
	Java		**Matlab**		**Python**	
	Value	n	Value	n	Value	n
1st	spiderman‡	20	spiderman‡	18	spiderman‡	9
2nd			6	2	5	2

‡ Trend identified across all three languages.

* Trend identified across two languages, but answer appeared in all groups.

◇ Trend identified across two languages.

Table 26: Common Incorrect Answers to Closed-Form Question Q23 Ranked by Frequency of Occurrence

variable: *array*1, 59.64% correct						
	Java		**Matlab**		**Python**	
	Value	n	Value	n	Value	n
1st	[1,2,3,4,4]‡	7	[1,2,3,4,4]‡	4	[1,1,3,4,5]	5
2nd	[0,2,3,4,5]	4	[1,2,3,4,9]	3	[1,2,3,4,4]‡	3
3rd	[1,2,4,4,5]‡	4	[1,2,4,4,5]‡	2	[1,2,4,4,5]‡	2
4th	[1,2,3,4,3]	2			[1,3,5,7,9]	2
5th	[1,2,5,4,4]	2				

variable: *array*2, 52.73% correct						
	Java		**Matlab**		**Python**	
	Value	n	Value	n	Value	n
1st	[9,14,5,6,8]*	8	[8,12,3,6,8]	7	[9,14,5,6,8]*	5
2nd	[9,12,4,6,8]	2	[5,12,3,6,8]	4	[0,2,4,6,8]	2
3rd	[9,13,4,6,8]◇	2	[9,3,3,6,8]	3		
4th	[9,14,4,6,8]	2	[9,11,3,6,8]	3		
5th			[9,13,4,6,8]◇	3		

‡ Trend identified across all three languages.
* Trend identified across two languages, but answer appeared in all groups.
◇ Trend identified across two languages.

Table 27: Common Incorrect Answers to Closed-Form Question Q26 Ranked by Frequency of Occurrence

variable: *returnvalue*, 50.55% correct				
	Matlab		**Python**	
	Value	n	Value	n
1st	4	4	$tryMe('ississippi','i',2)$‡	4
2nd	11	3	2‡	2
3rd	$tryMe('ississippi','i',2)$‡	3		
4th	2‡	2		
5th	22	2		
6th	44	2		

‡ Trend identified across all languages.

APPENDIX C

PSEUDO-CODE GUIDE

This appendix contains the pseudo-code overview that was provided to participants for the FCS1 Assessment studies described in Chapter 4.

Pseudo-code Guide

Syntax	Examples
Statements *variable = expression* Math Operators: +, -, *, /, % Relational Operators: ==, <, <=, >, >=, != Conditional Operators: AND, OR, NOT Booleans: True/False	total = 0 x = 4 * 3 name = "John Smith" flag = True answer = 1 AND (1 OR 0)
Print # print without new line PRINT *value, value* # print value with new line PRINTLN value	**Output** PRINT "Hello" ➡ Hello World PRINTLN "World" PRINTLN "Hello","World" ➡ Hello World
If Statement (Conditional) IF *condition* THEN *statement(s)* …… ELSE IF *condition* THEN *statement(s)* ELSE *statement(s)* ENDIF	IF testScore >= 90 THEN grade = 'A' ELSE IF testScore >= 80 THEN grade = 'B' ELSE IF testScore >= 70 THEN grade = 'C' ELSE grade = 'F' ENDIF
For Loop (Definite) # counter from *start-value* up to but not # including *end-value* FOR *counter = start-value* TO *end-value* BY # DO *statement(s)* …… ENDFOR	FOR x = 1 to 5 BY 1 DO xSquared = x * x PRINTLN x, xSquared ENDFOR ➡ **Output** 1 1 2 4 3 9 4 16
While Loop (Indefinite) WHILE *condition* DO *statement(s)* …… ENDWHILE	count = 5 WHILE count < 9 DO PRINTLN count count = count + 1 ENDWHILE ➡ **Output** 5 6 7 8
Functions DEFINE *function-name(parameter, parameter, …)* *statement(s)* …… RETURN *value* ENDDEF	DEFINE findMax(num1, num2) IF num1 >= num2 THEN maxNum = num1 ELSE maxNum = num2 ENDIF RETURN maxNum ENDDEF

Pseudo-code Guide

Classes and Objects

```
CLASS class-name

    #constructor used to initialize values
    DEFINE initialize class-name()
        variable = initial value
    ENDDEF

    DEFINE function-name(parameter, …)
        statement(s)
    ENDDEF
ENDCLASS
```

Creating Objects:

```
object-name = new class-name
```

Method calls:

```
object-name.function-name()
```

```
CLASS Car
    DEFINE initialize Car()
        wheels = 4
        currentSpeed = 100
    ENDDEF

    DEFINE speedUp(newSpeed)
        currentSpeed = newSpeed
    ENDDEF
ENDCLASS
```

Creating Objects:

```
car1 = new Car()
```

Method calls:

```
car1.speedUp(200)
```

String and Arrays/Lists

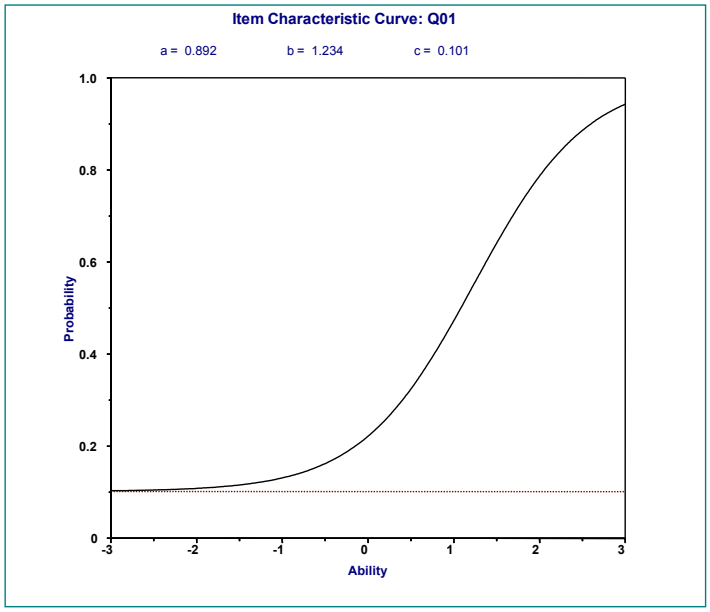
```
#length of string
length(string variable)

#convert to uppercase/lowercase
string variable.upper()
string_variable.lower()

#find index of 1st instance of substring
string_variable.find(substring)
#return number of substrings in string
string_variable.count(substring)

# string splicing, exclusive
# include indexes of string from start up to,
# but not including, end
string variable.substring(start : end)

list_variable = [item1, item2, item3, …]

#length of array/list
length(list_variable)

# access the first element
list variable[0]

#add/delete element at position index
add(list variable[index], value)
del(list_variable[index])
```

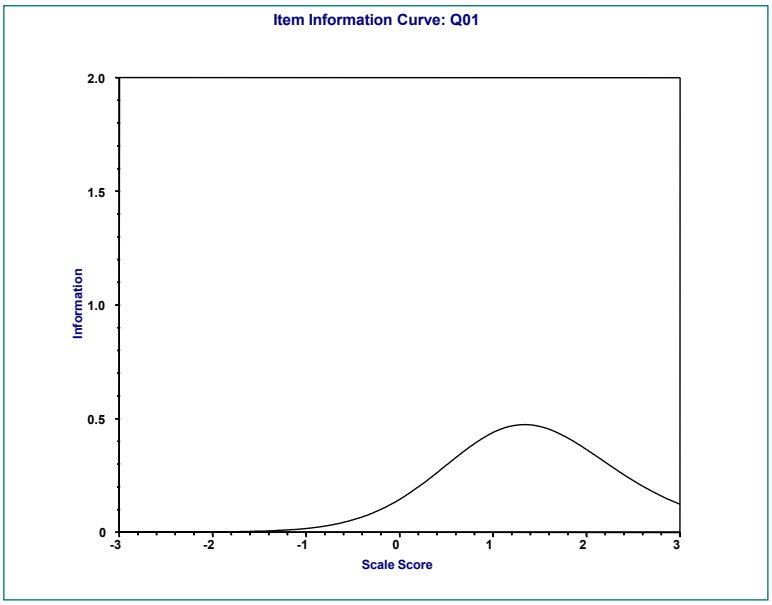
```
name = "John Smith"                    Output
length(name)                             10

name.upper()                    ➡   JOHN SMITH
name.lower()                        john smith

name.find('Sm')                          5
name.count('h')                          2

name.substring(2 : 6)                 "hn S"

sports = ['soccer','football','hockey']
                                       Output
length(sports)                           3

sports[0]                       ➡    'soccer'

add(sports[3], 'baseball')      ➡
sports = ['soccer','football','hockey',
'baseball']

del(sports[2])                  ➡
sports = ['soccer','football',
'baseball']
```

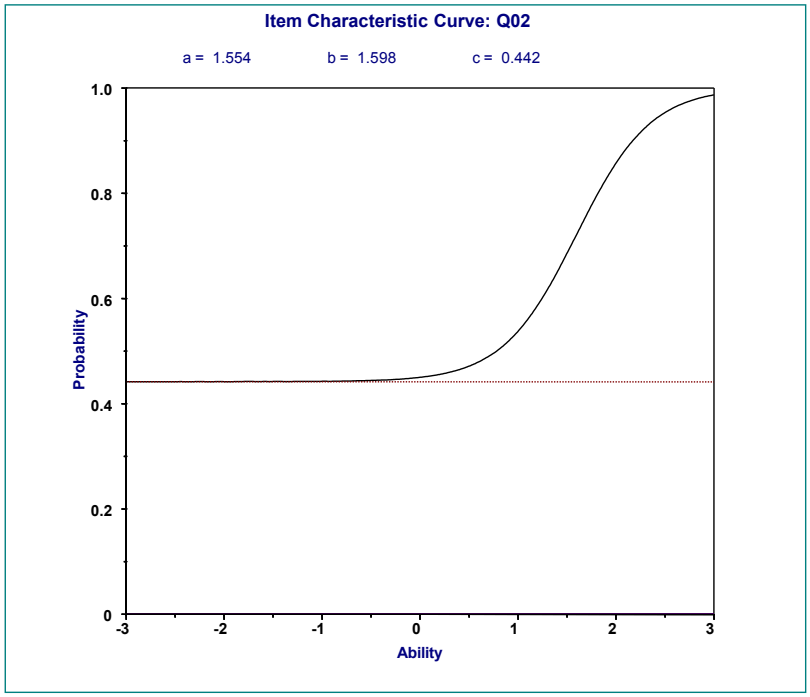
APPENDIX D

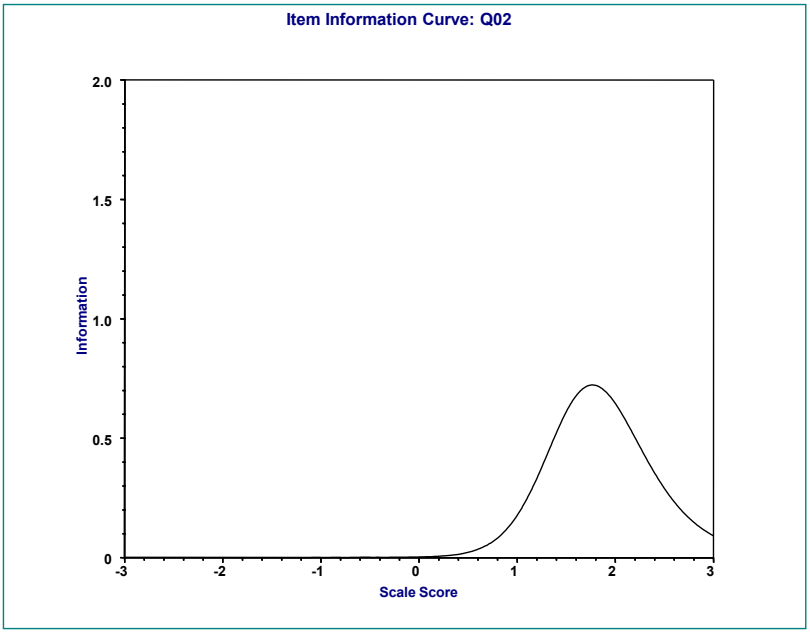
ITEM RESPONSE THEORY ANALYSIS

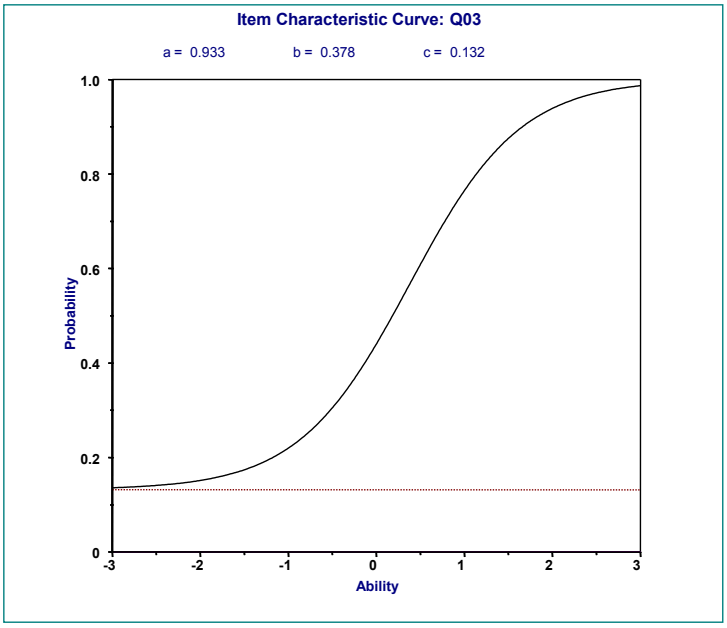
This appendix contains the item characteristic curves (ICC) and item information curves (IIC) for the FCS1 Assessment questions, as discussed in Chapter 5.

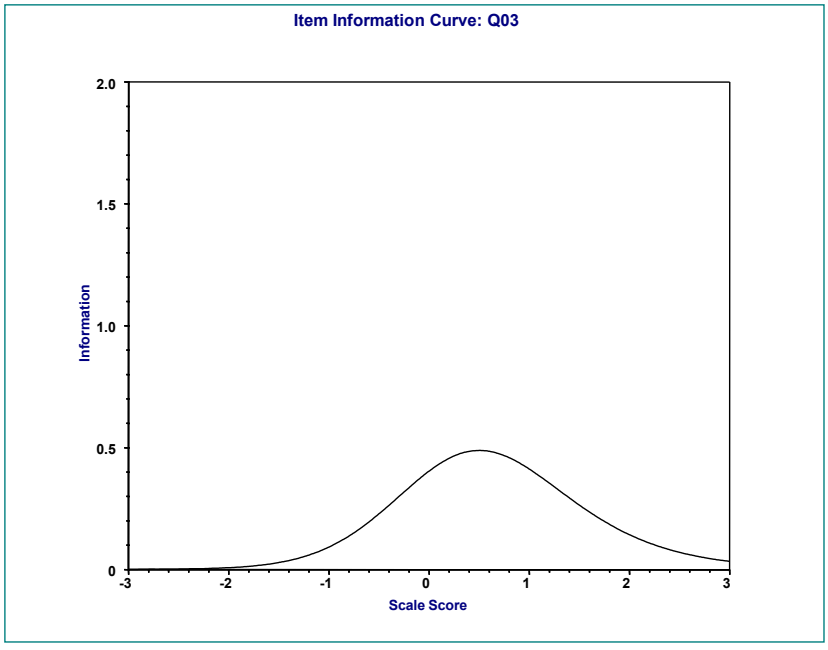


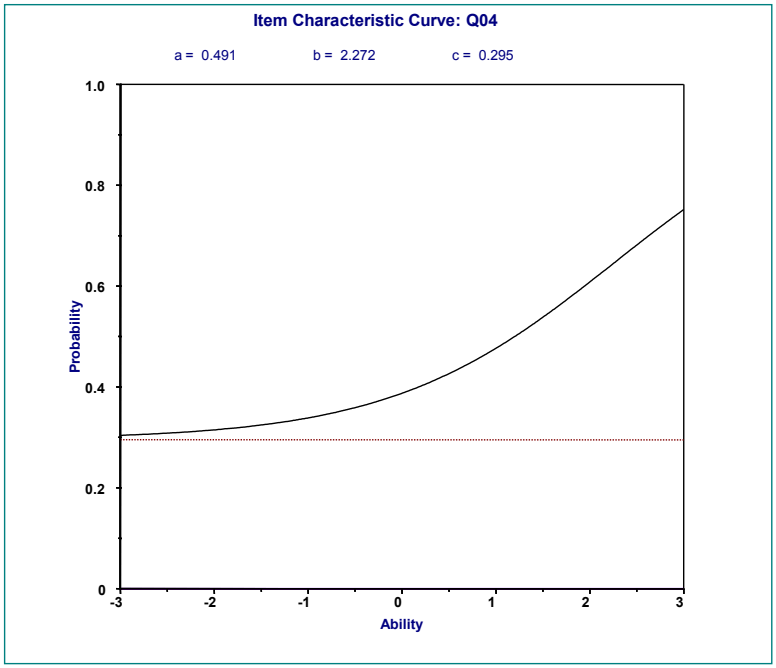


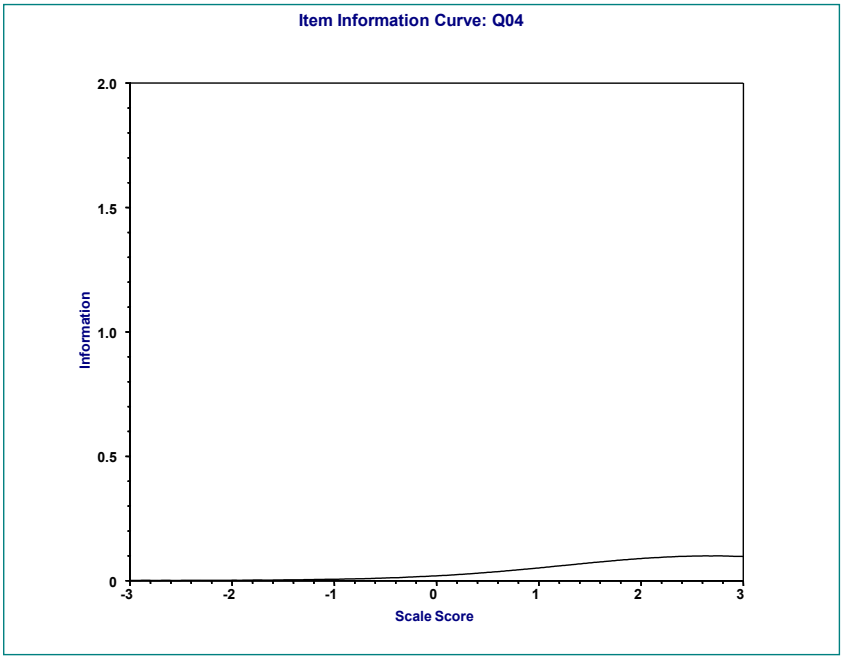


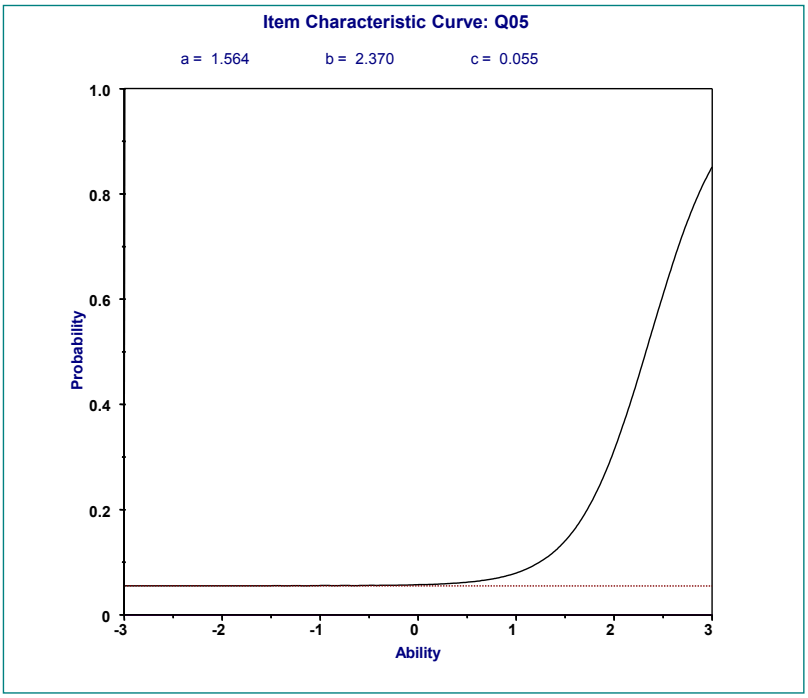


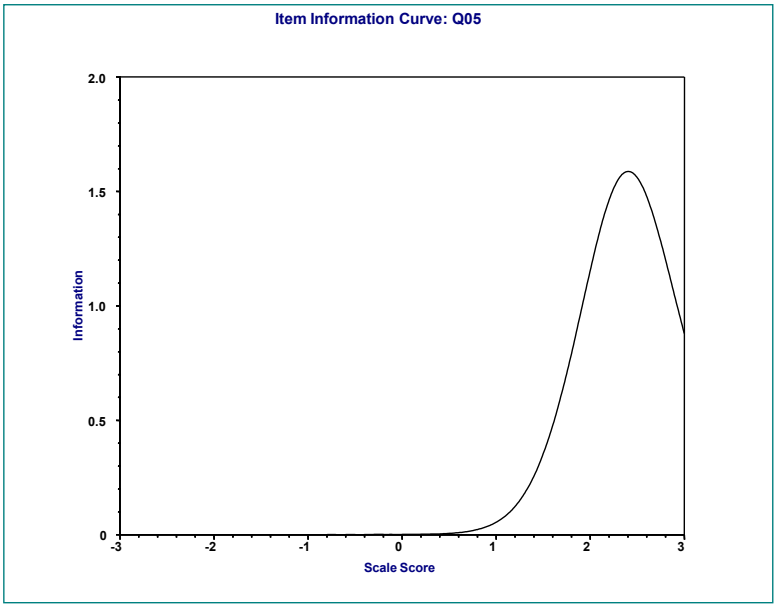


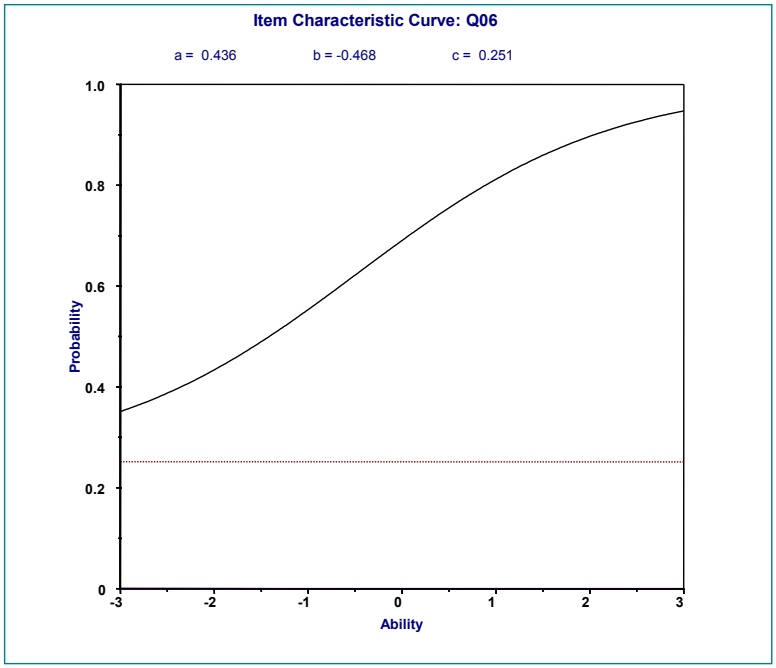


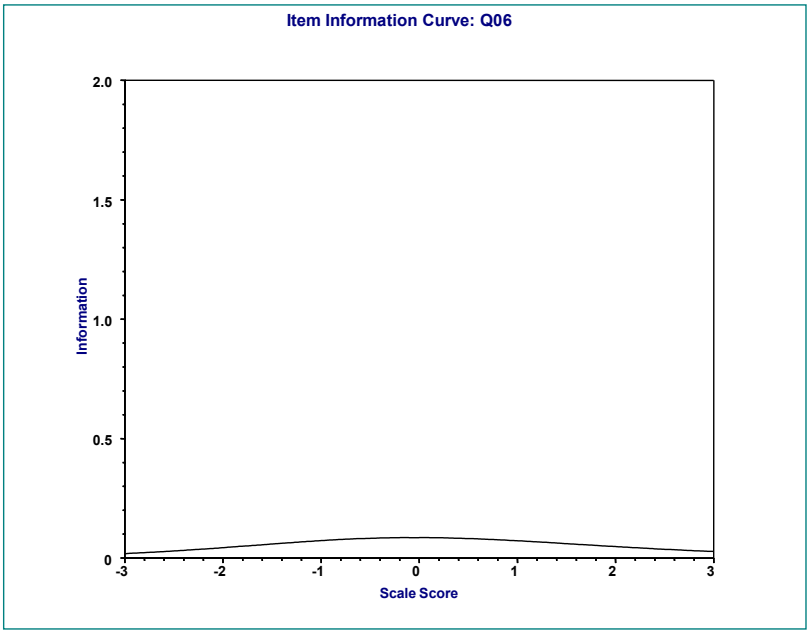


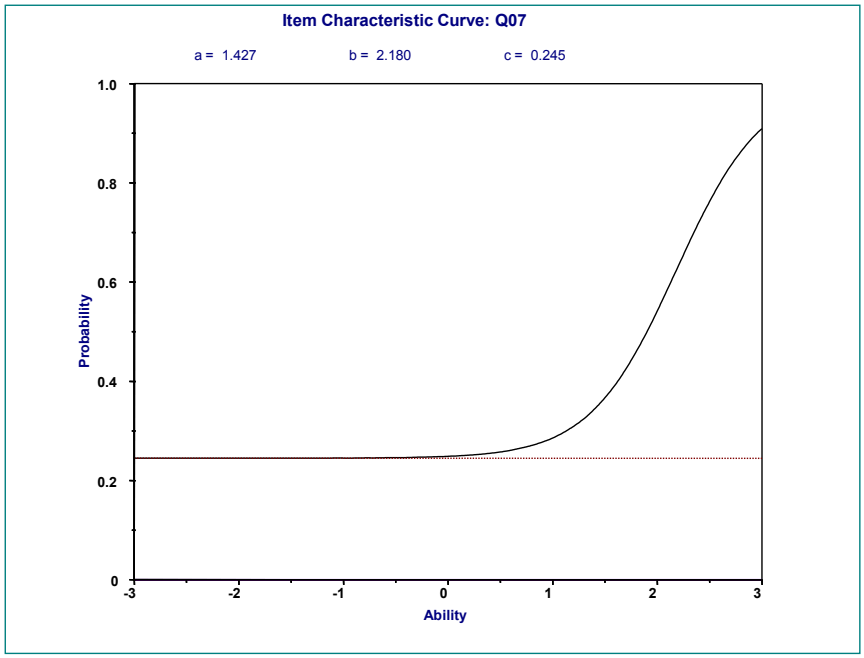


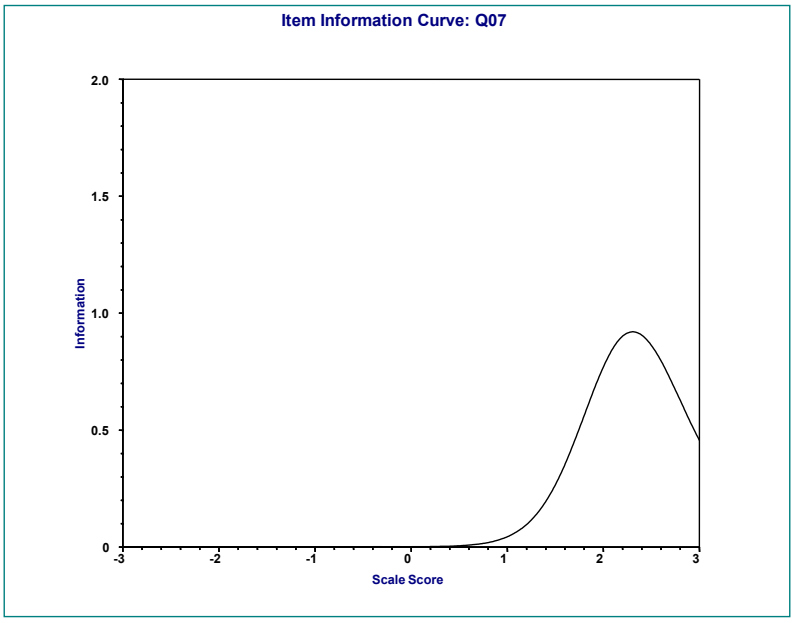


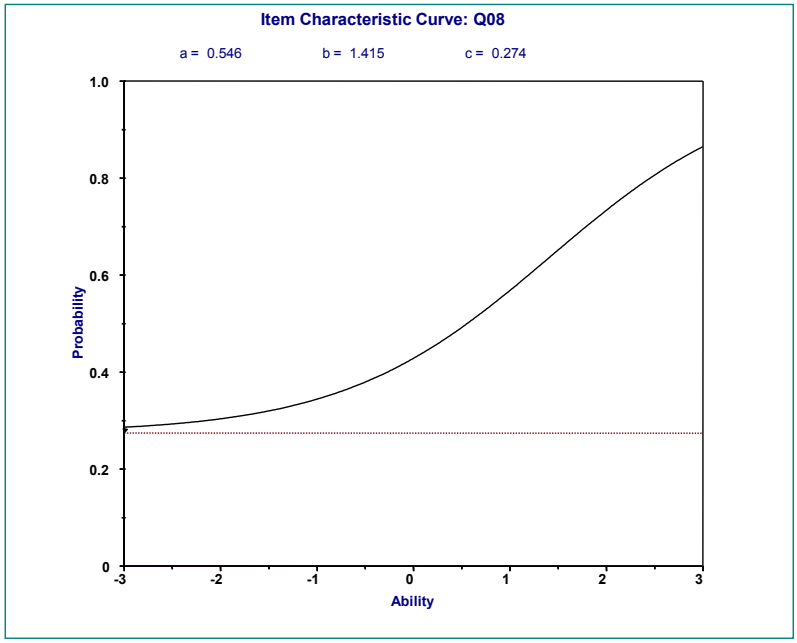


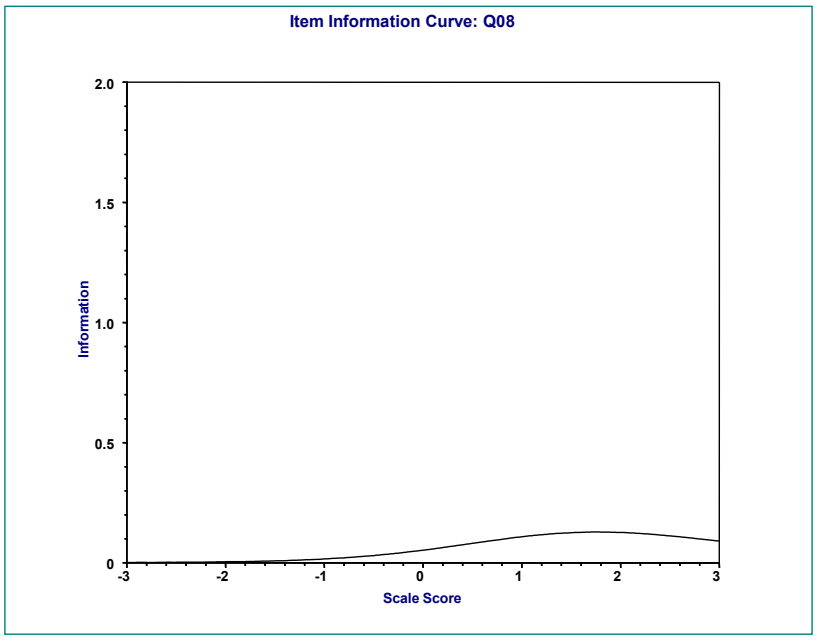


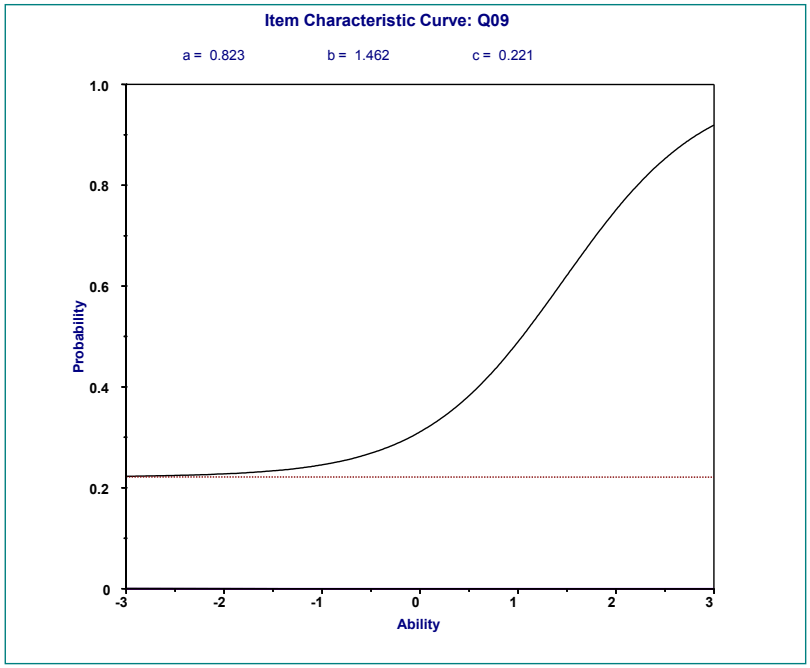


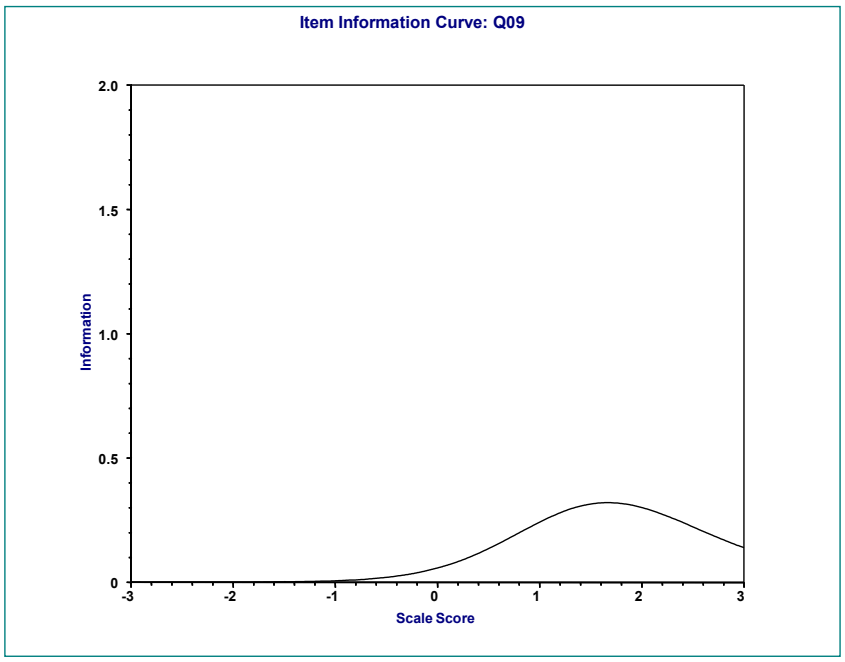


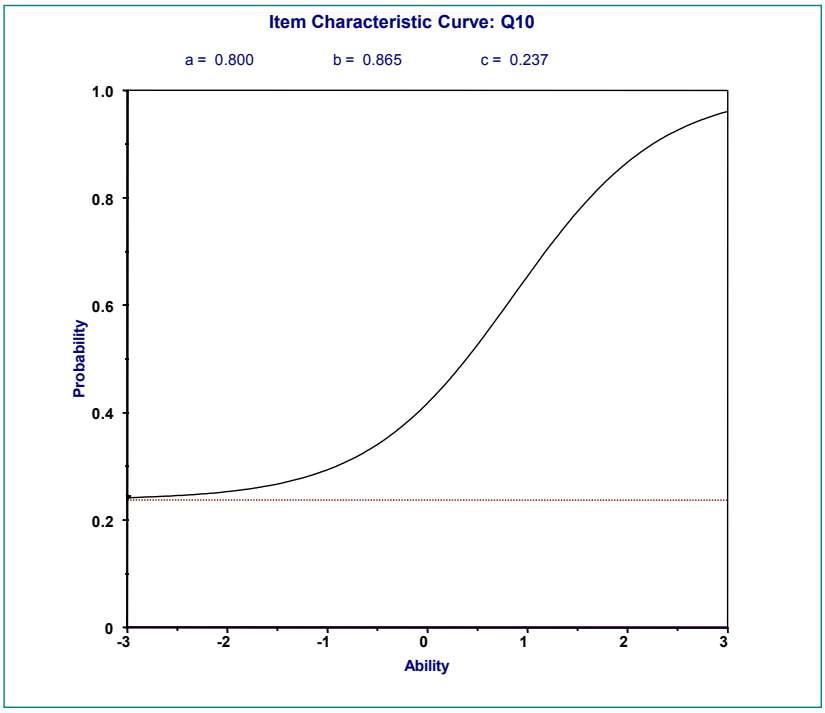


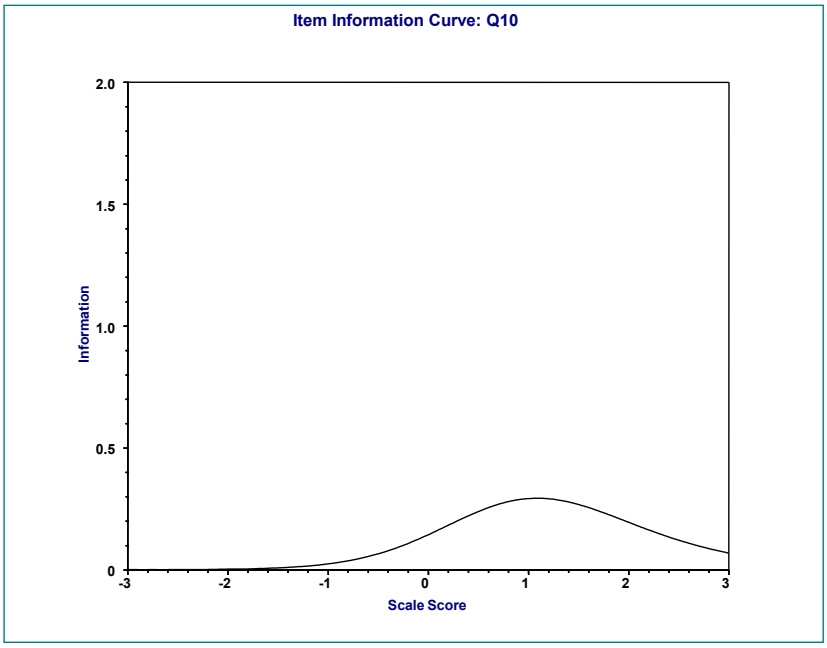


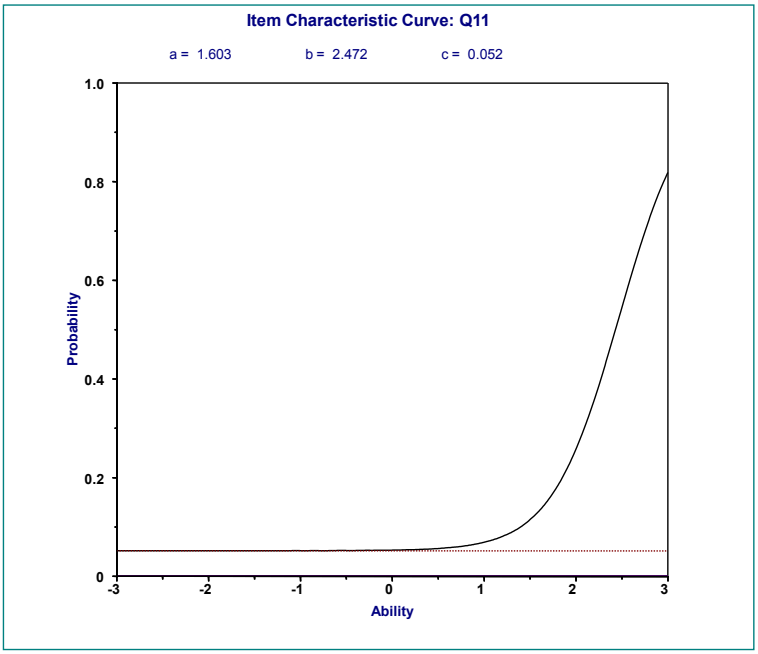


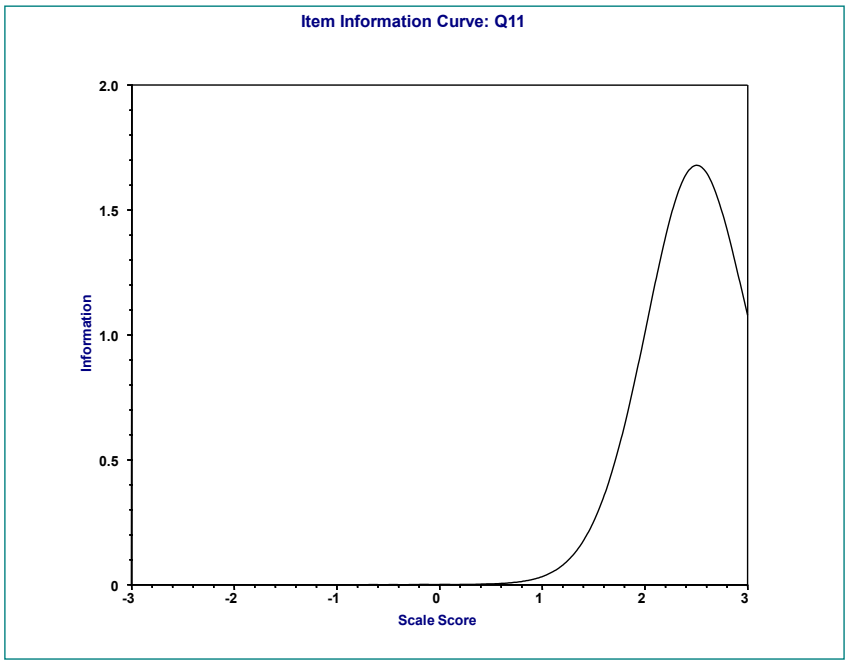


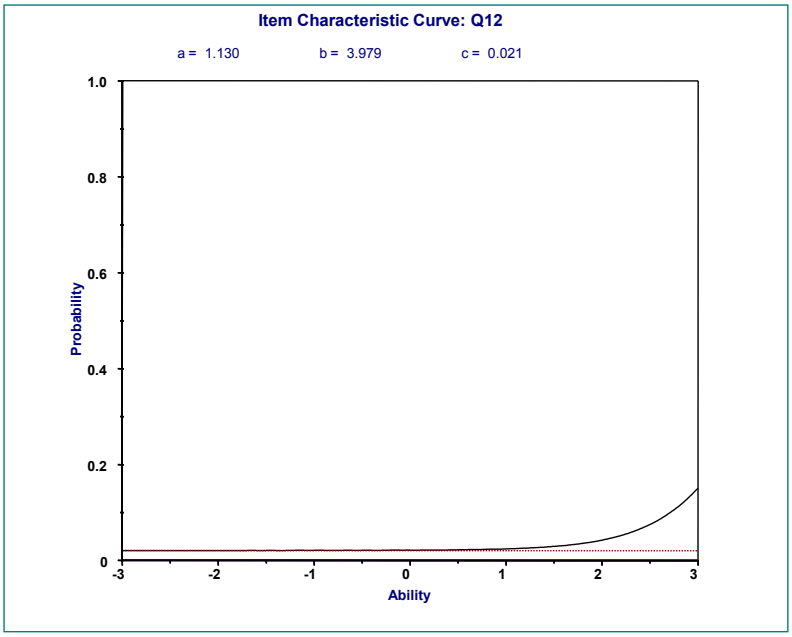


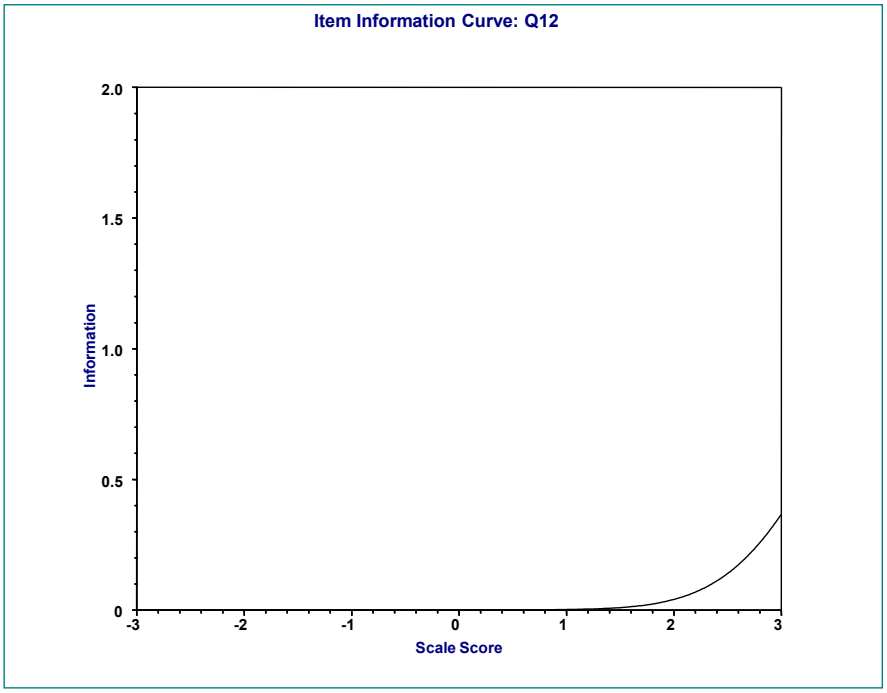


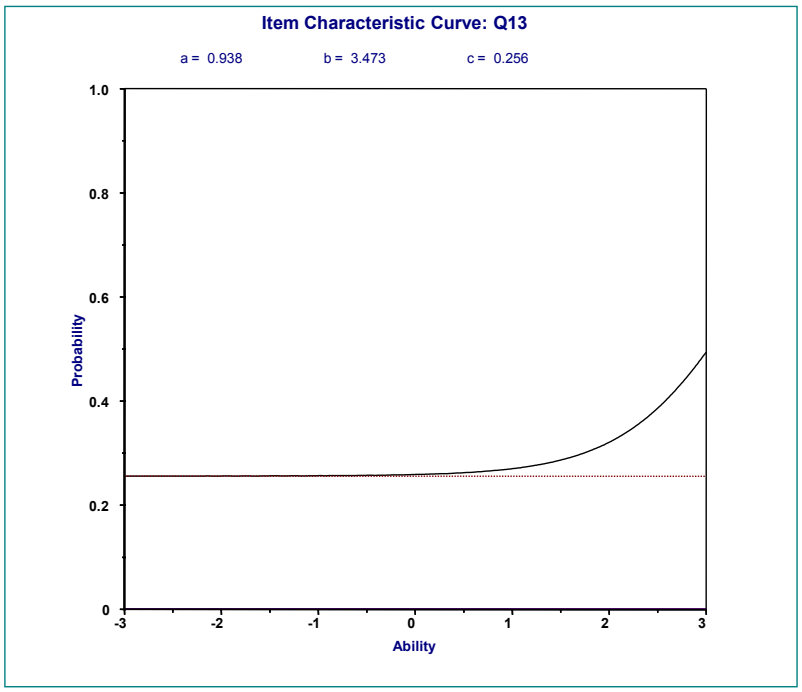


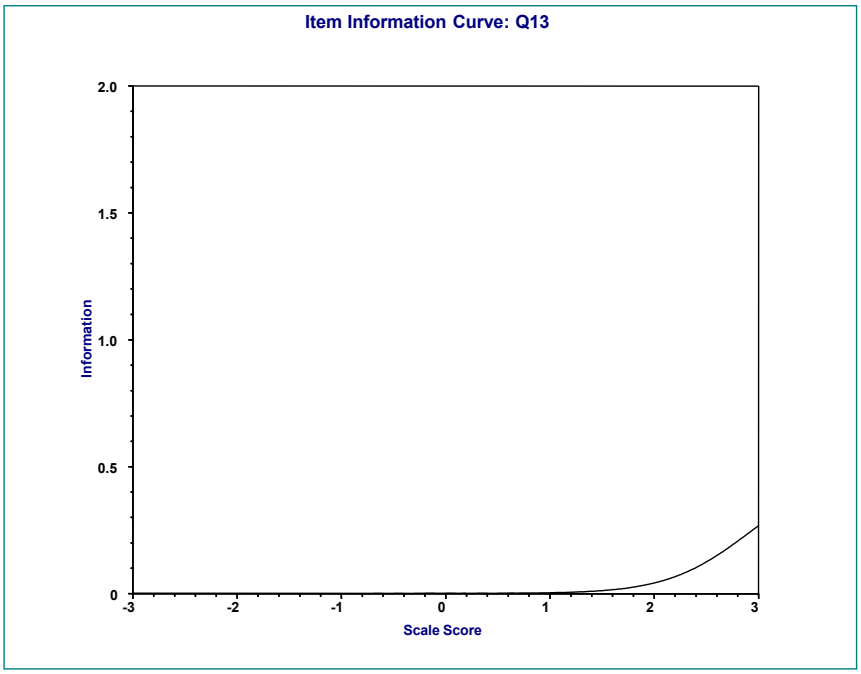


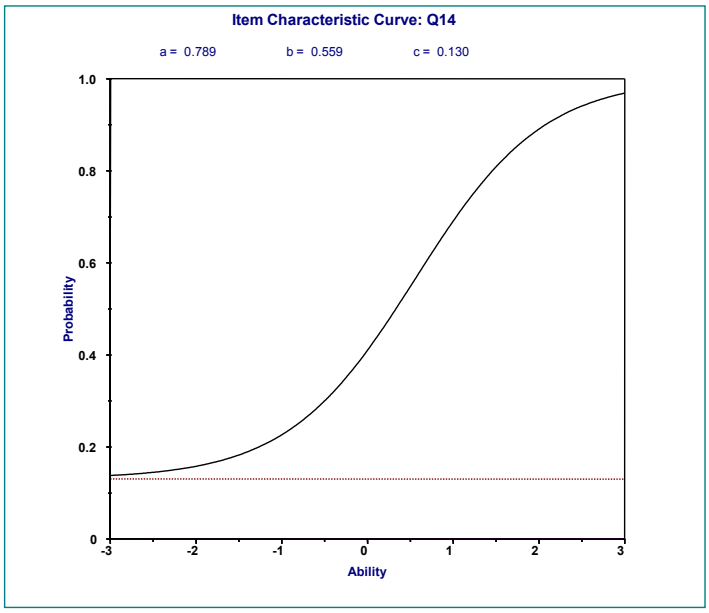


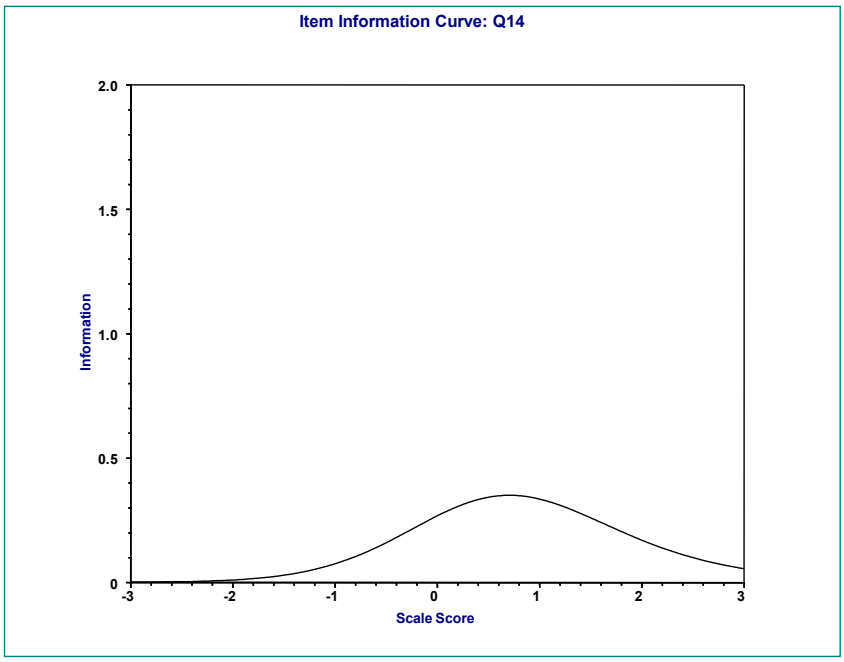


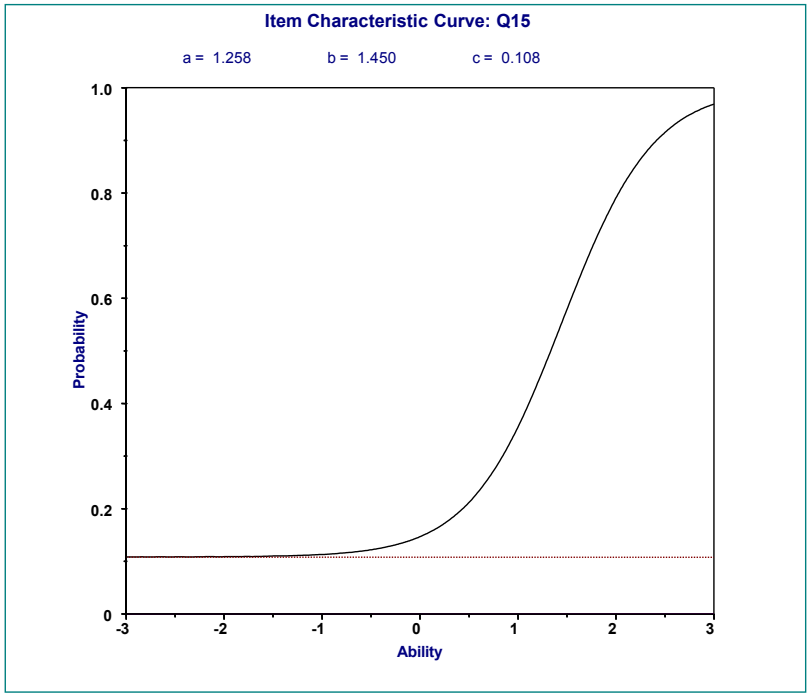


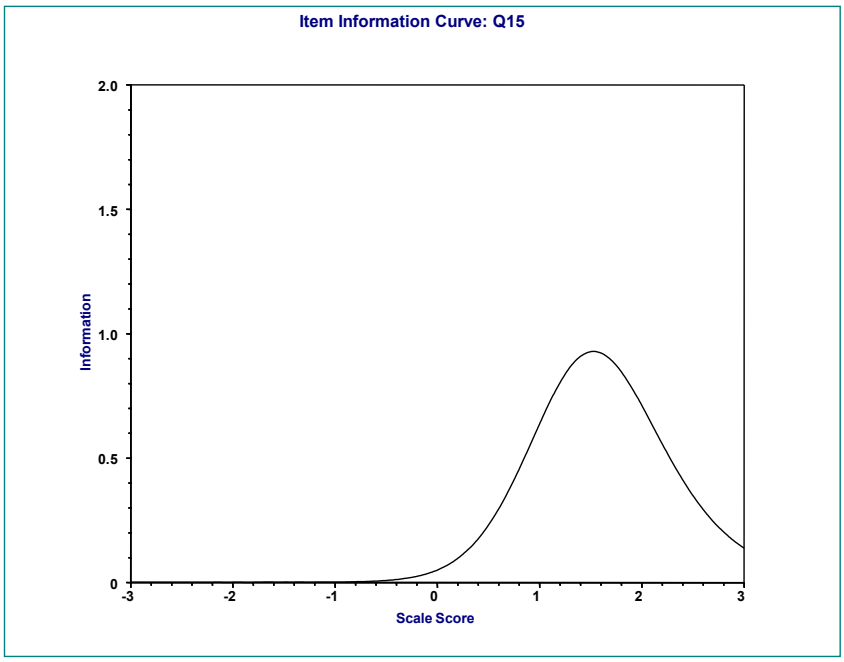


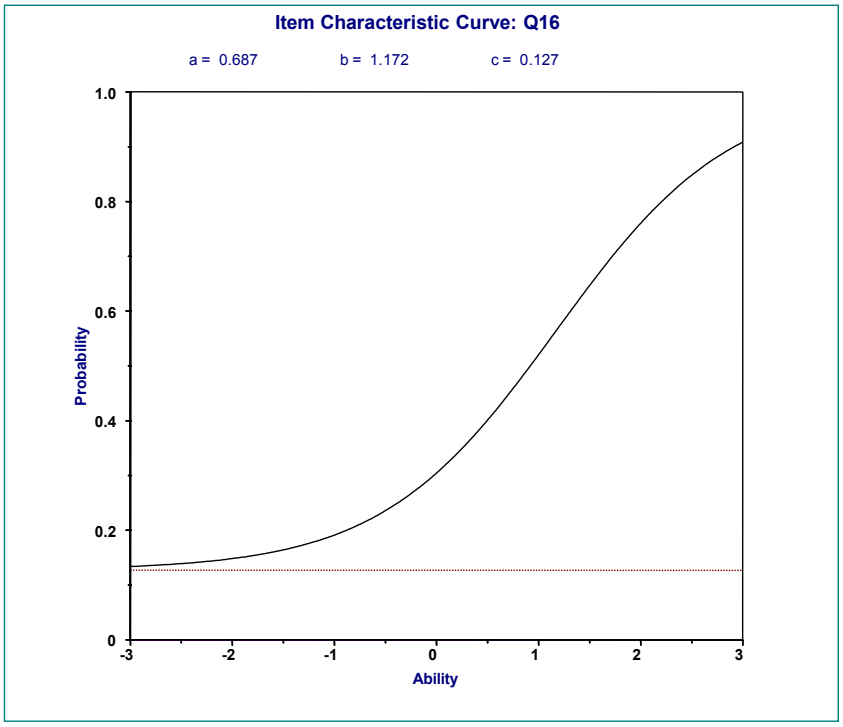


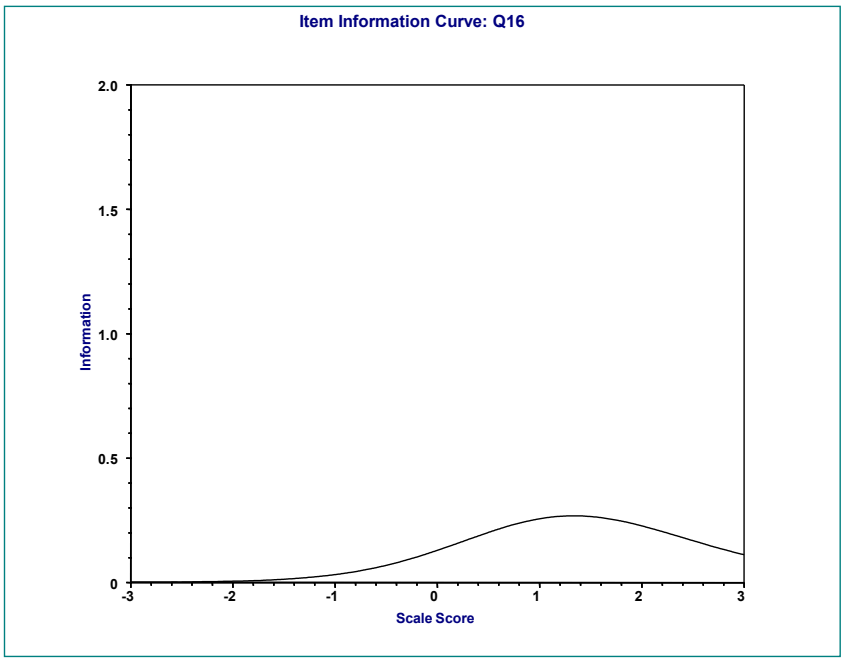


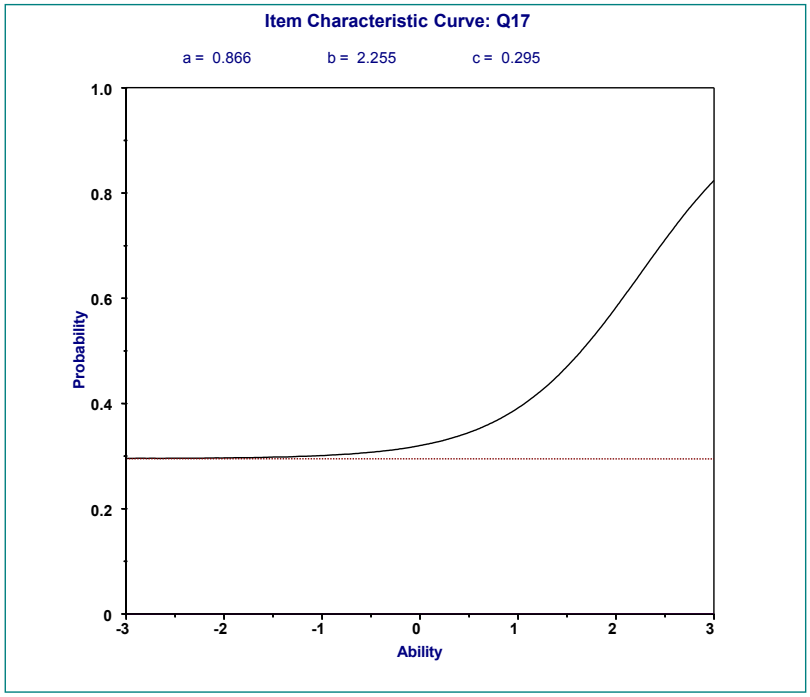


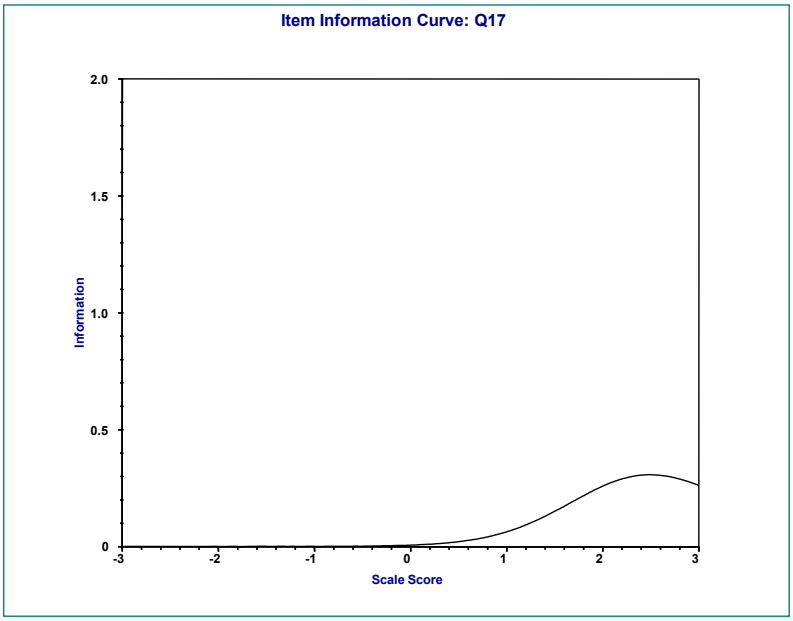


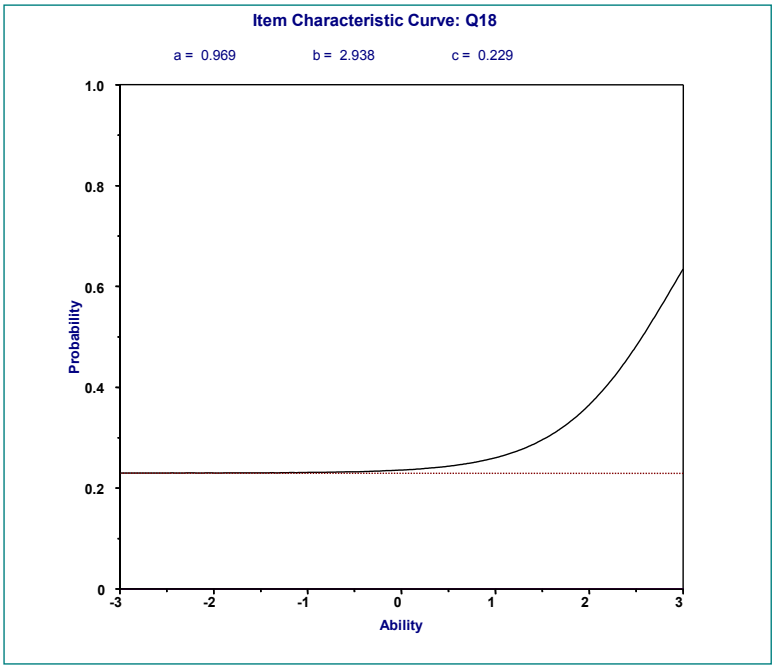


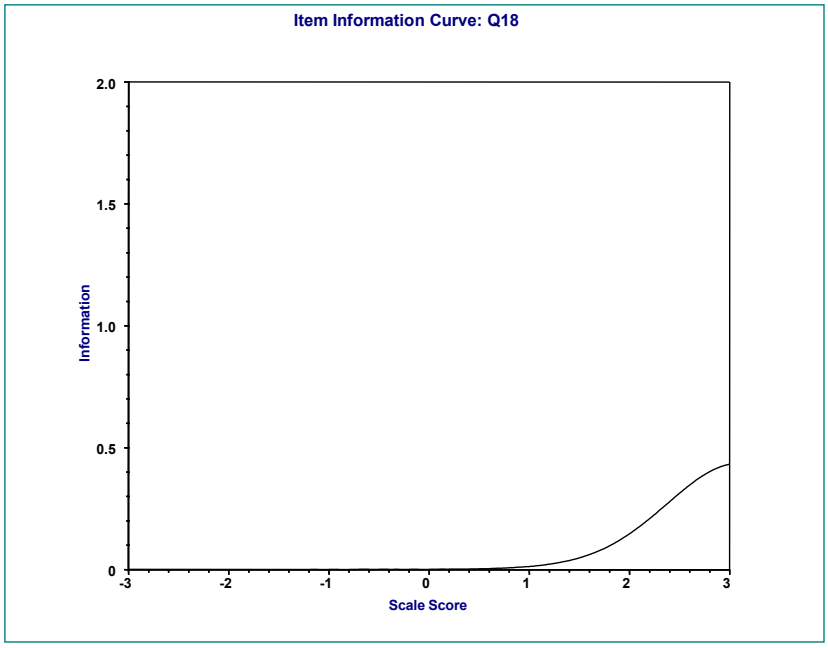


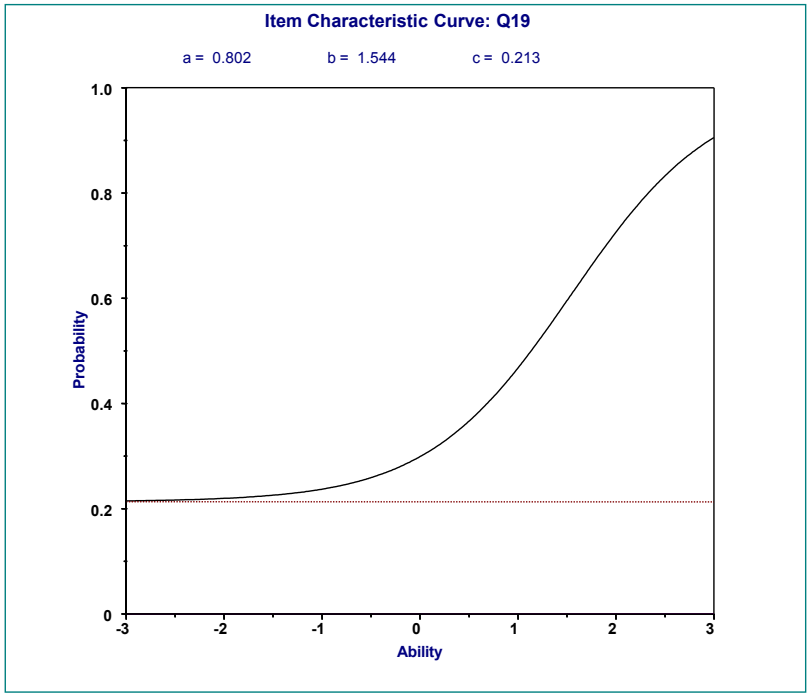


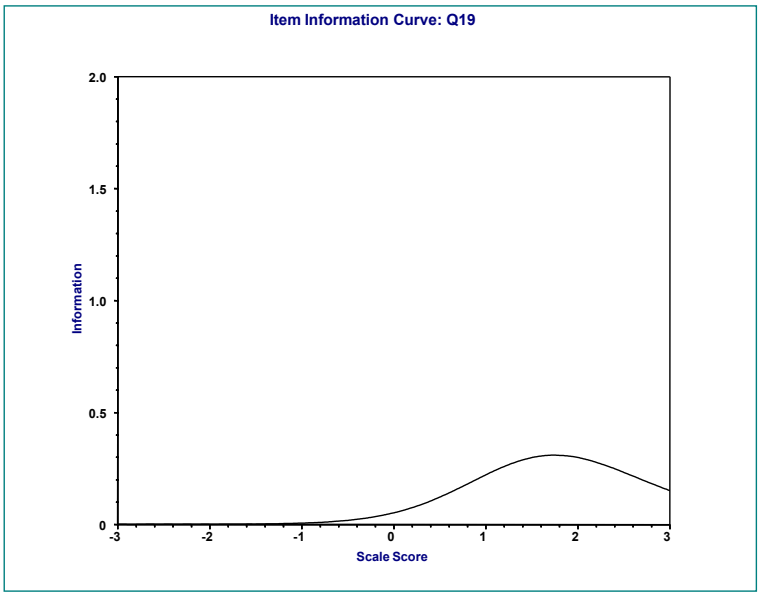


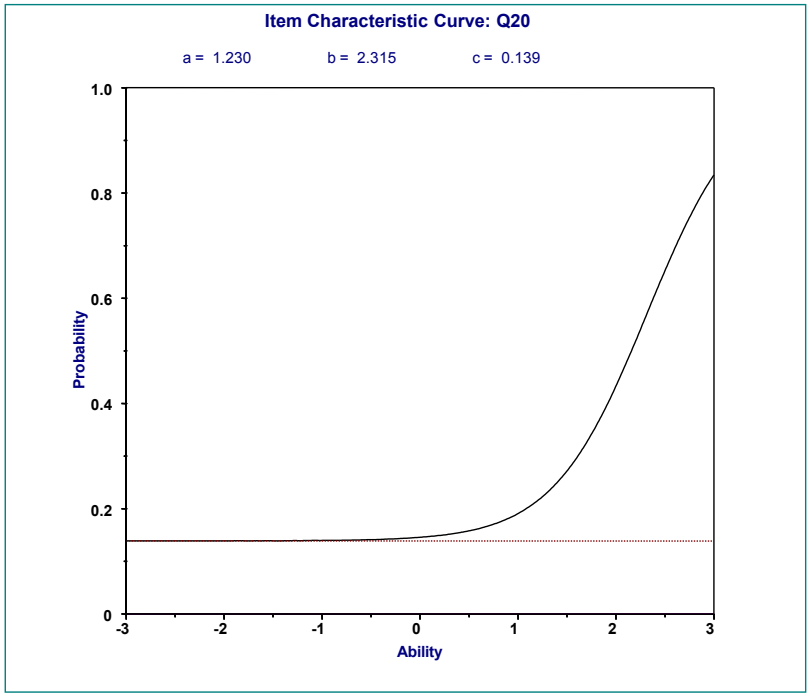


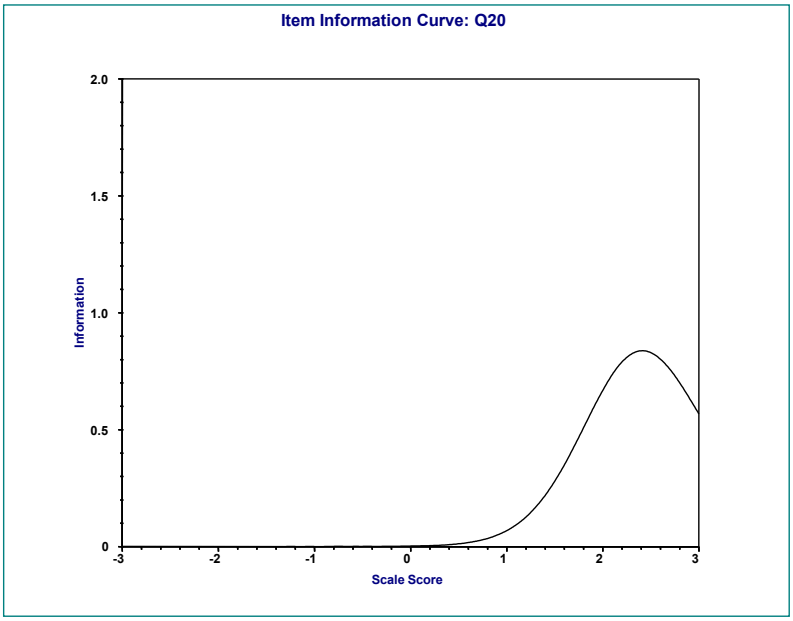


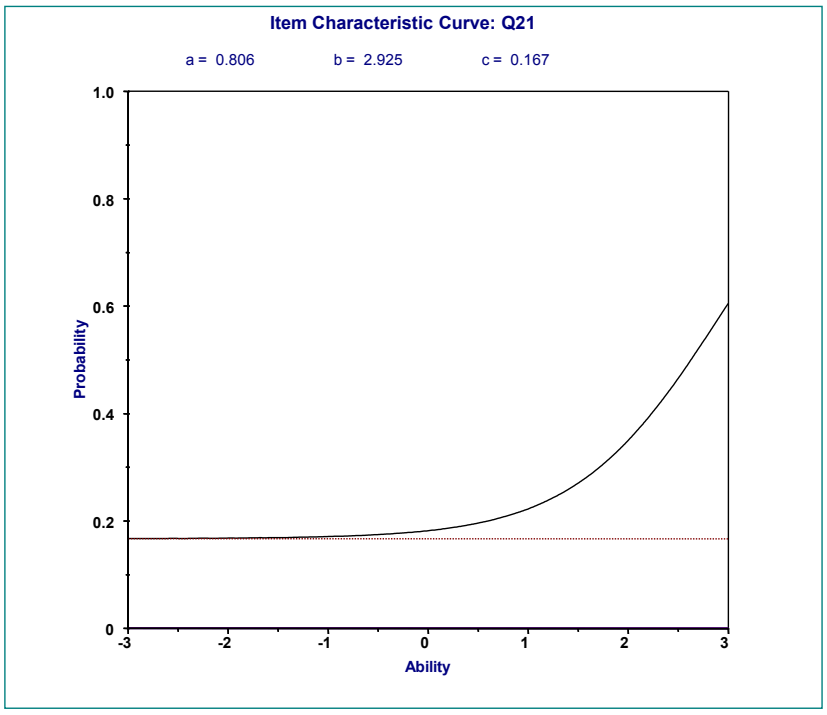


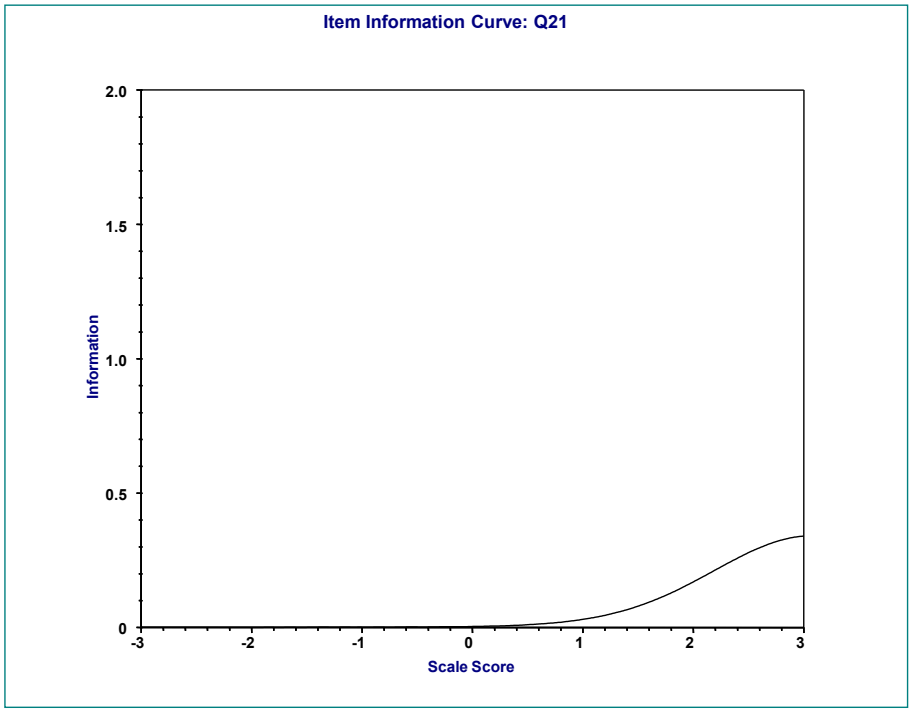


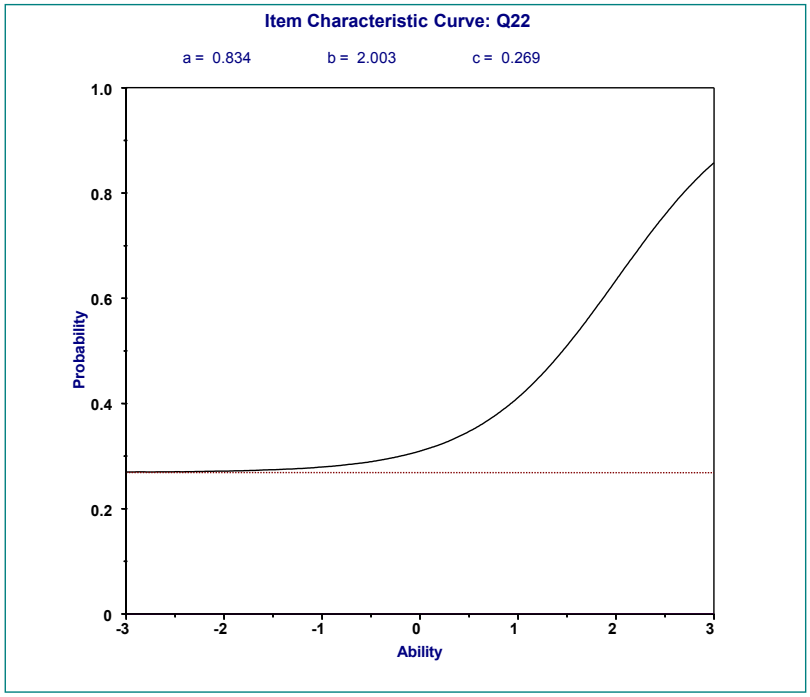


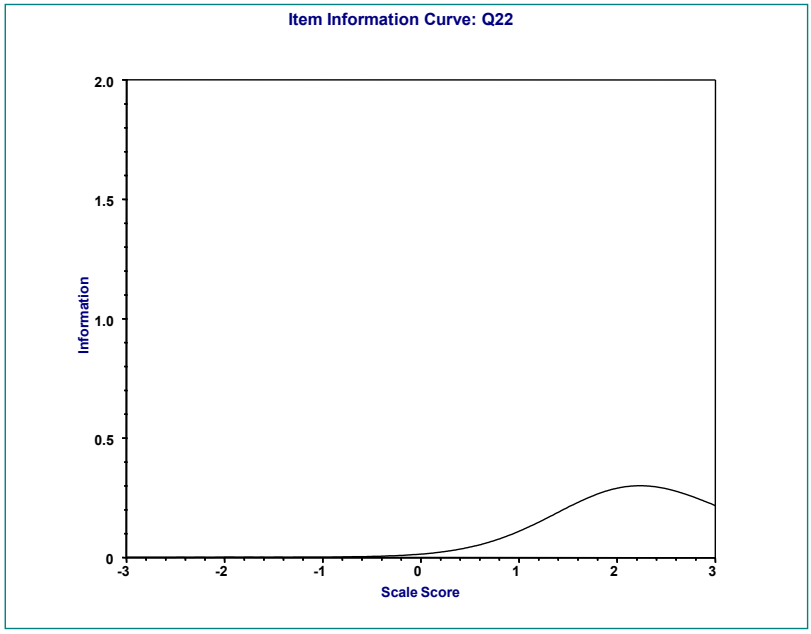


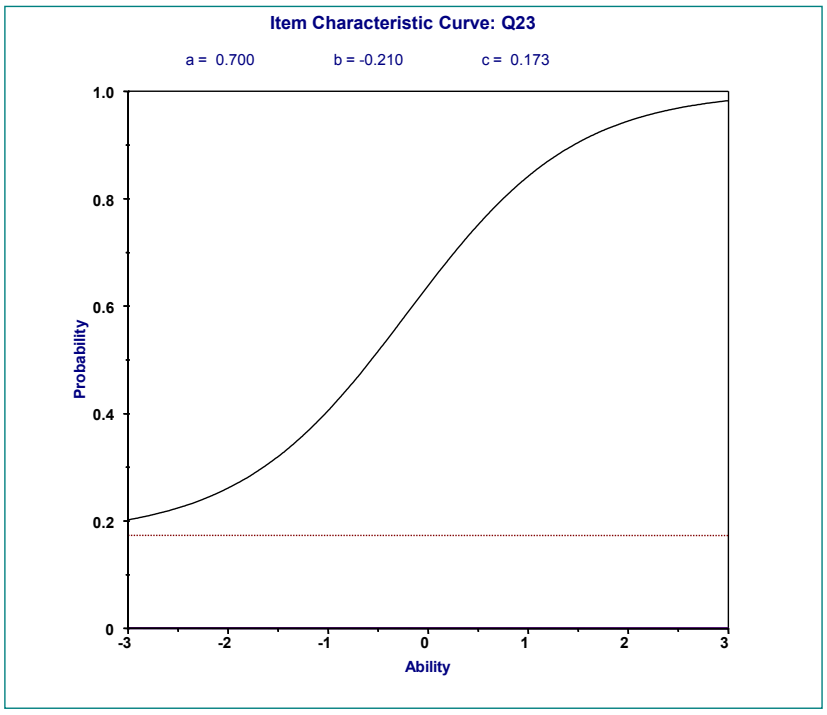


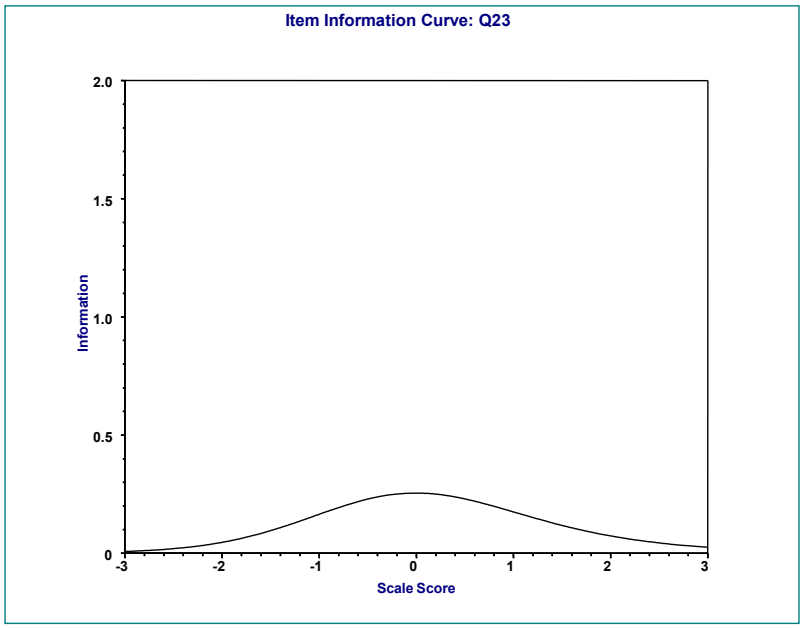


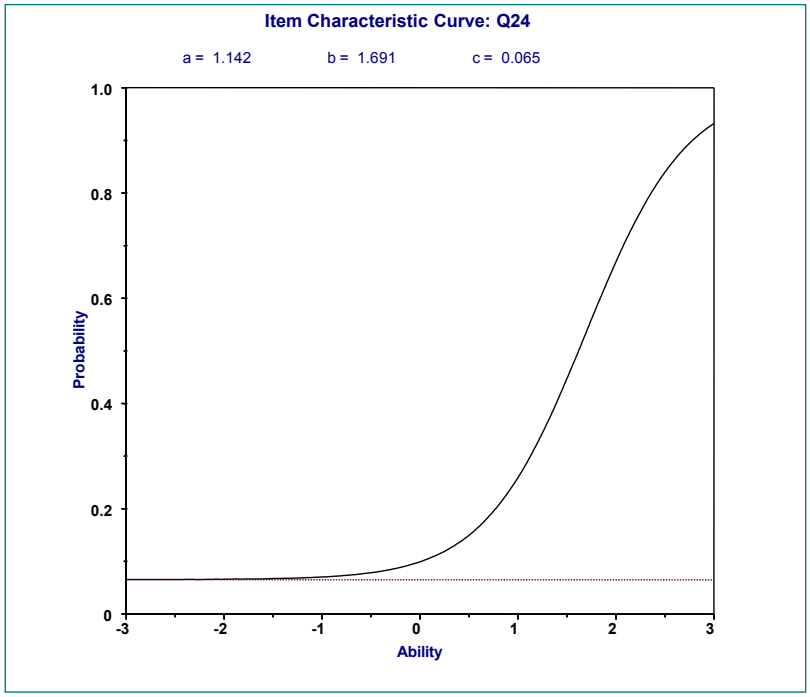


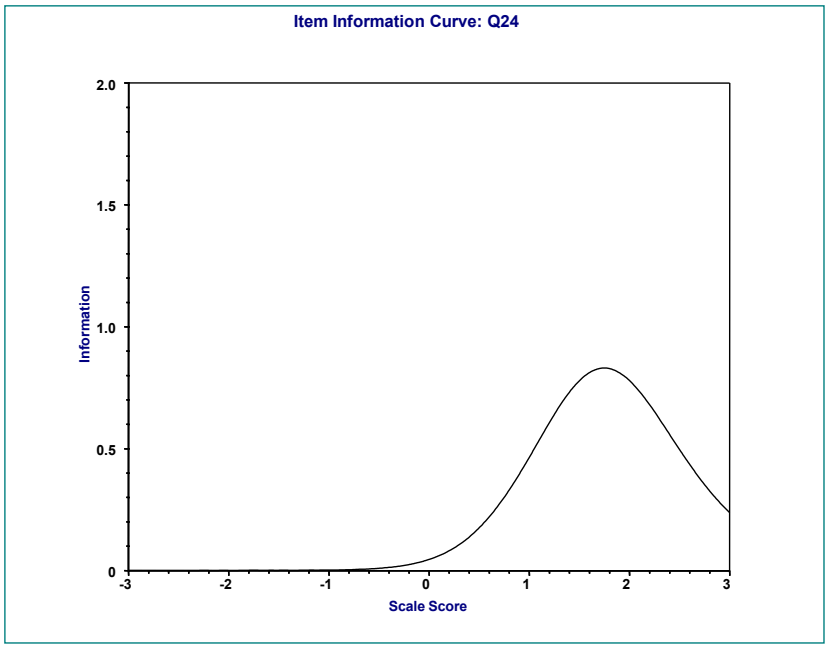


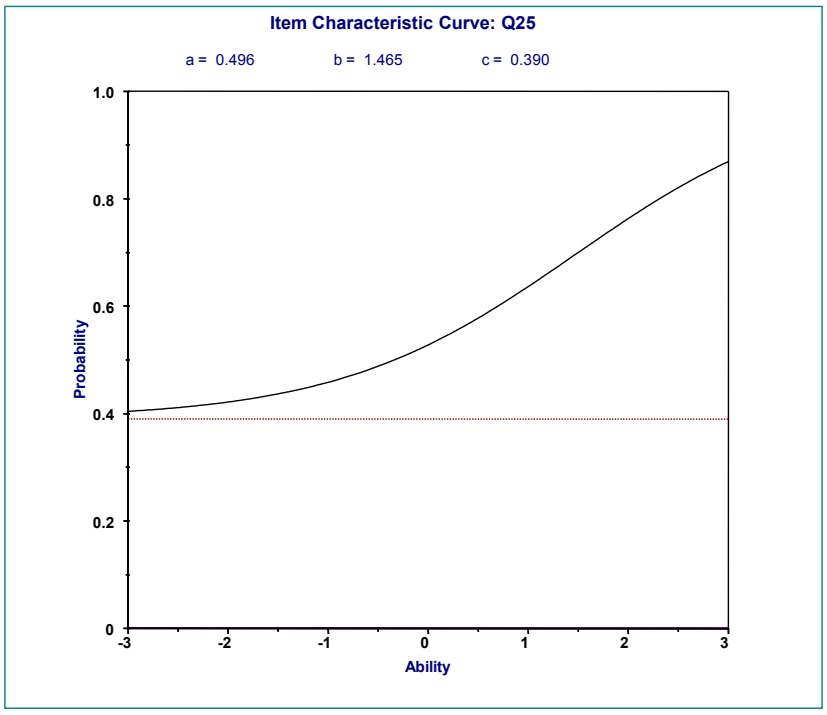


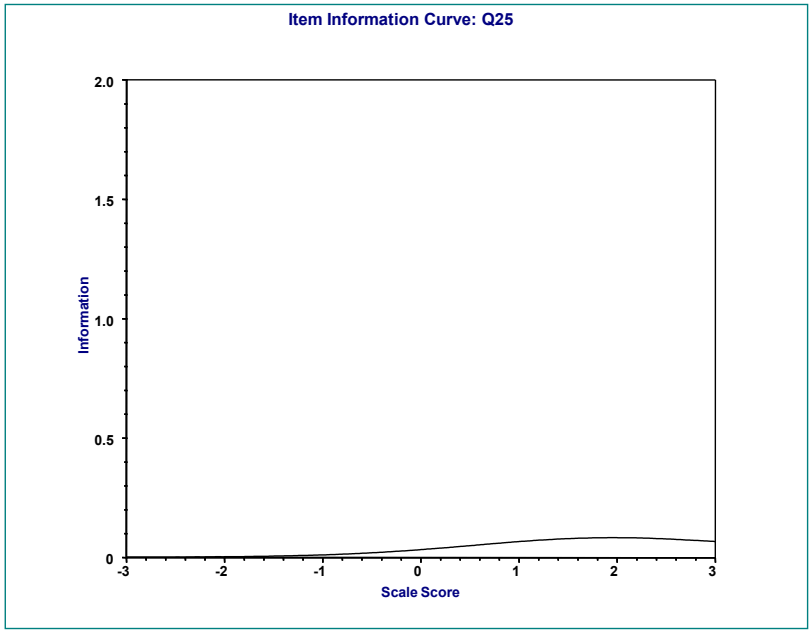


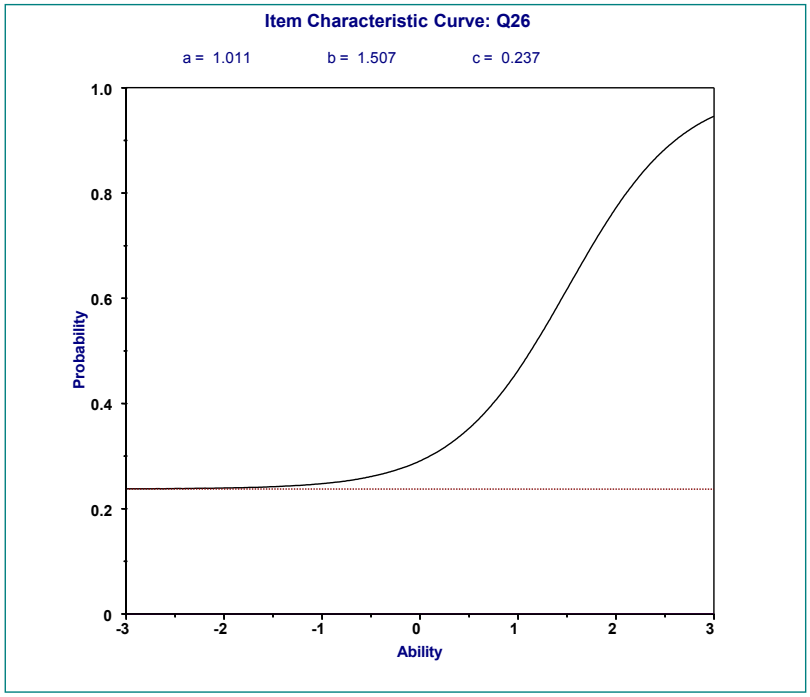


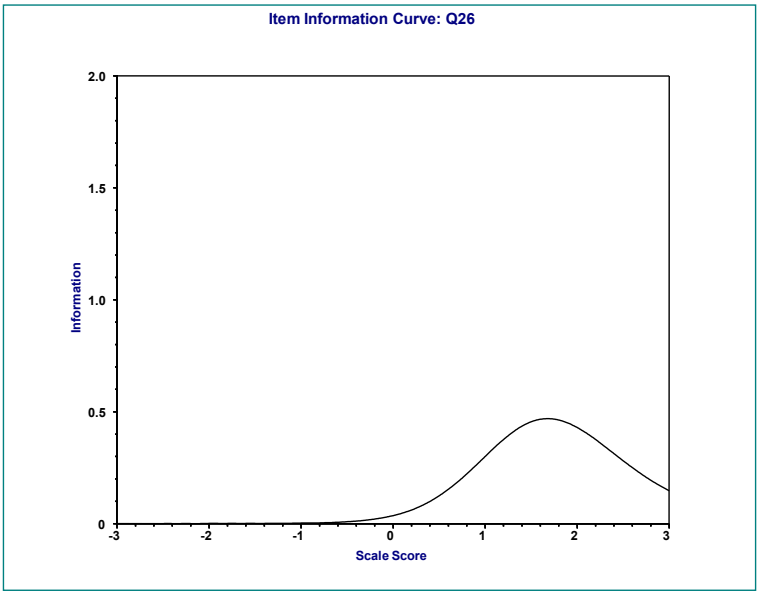


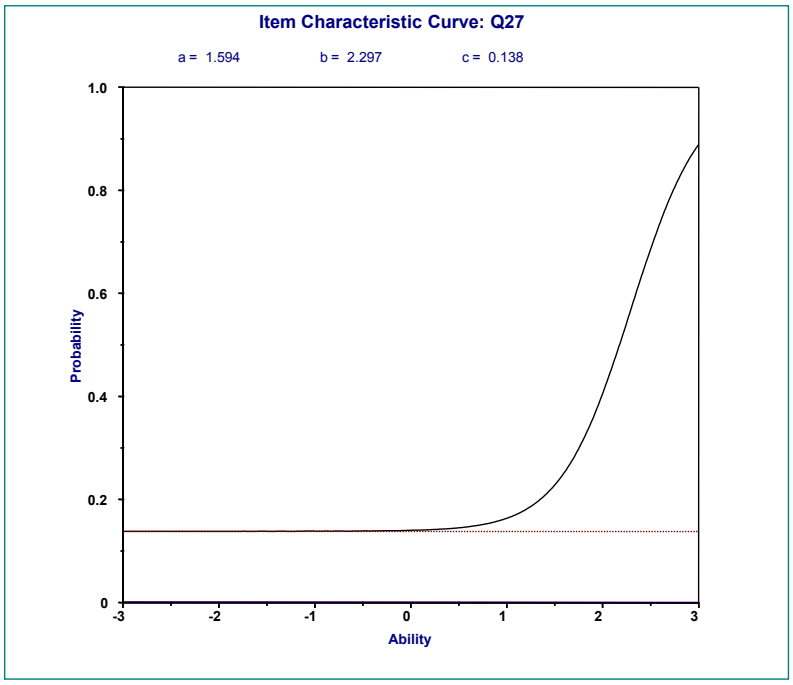


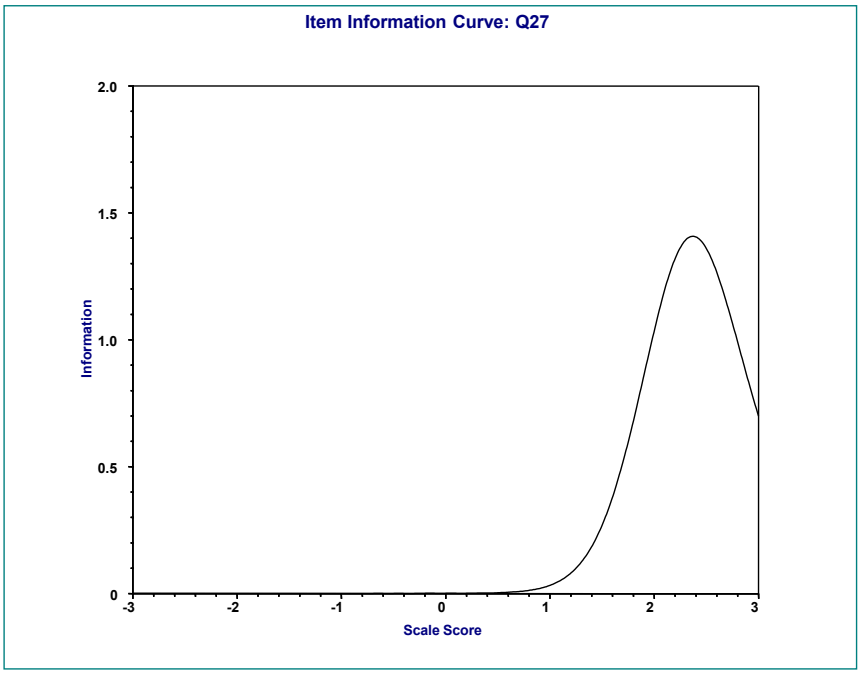












REFERENCES

Adams, W. K., Perkins, K. K., Podolefsky, N. S., Dubson, M., Finkelstein, N. D., & Wieman, C. E. (2006). New instrument for measuring student beliefs about physics and learning physics: The Colorado learning attitudes about science survey. *Physical Review Special Topics – Physics Education Research*, *2*(1), 010101.

American Educational Research Association, American Psychological Association, & National Council on Measurement in Education. (1999). *Standards for educational and psychological testing*. Washington, DC: American Educational Research Association.

Baker, F. B. (2001). *The basics of item response theory* (2nd ed.). College Park, MD: ERIC Clearinghouse on Assessment and Evaluation.

Birnbaum, A. (1968). Some latent trait models and their use in inferring an examinee's ability. In F. M. Lord & M. R. Novick (Eds.), *Statistical theories of mental test scores* (p. 395-479). Reading, MA: Addison-Wesley.

Bradley, J. V. (1958). Complete counterbalancing of immediate sequential effects in a latin square design. *Journal of the American Statistical Association*, *53*(282), 525–528.

Braun, V., & Clarke, V. (2006). Using thematic analysis in psychology. *Qualitative Research in Pyschology*, *3*(2), 77–101.

Cohen, J. (1988). *Statistical power analysis for the behavioral sciences* (2nd ed.). Hillsdale, NJ: Lawrence Erlbaum Associates.

Cohoon, J., & Davidson, J. (2006). *Java 5.0 program design*. Boston, MA: McGraw Hill.

College Level One Team. (n.d.). *Field-tested learning assessment guide* [web page]. `http://www.flaguide.org/`.

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

Dann, W., Cooper, S., & Pausch, R. (2006). *Learning to program with Alice*. Upper Saddle River, NJ: Prentice Hall.

Decker, A. M. (2007). *How students measure up: An assessment instrument for introductory computer science*. Unpublished doctoral dissertation, University at Buffalo (SUNY), Buffalo, NY.

Deitel, H., & Deitel, P. (2005). *C++: How to program* (5th ed.). Upper Saddle River, NJ: Prentice Hall.

Ding, L., Chabay, R., Sherwood, B., & Beichner, R. (2006). Evaluating an electricity and magnetism assessment tool: Brief electricity and magnetism assessment. *Physical Review Special Topics – Physics Education Research*, *2*(1), 010105.

Driscoll, M., & Bryant, D. (1998). *Learning about assessment, learning through assessment*. Washington, D.C.: National Academy Press.

Epstein, J. (1997). Cognitive development in an integrated mathematics and science program. *Journal of College Science Teaching*, *27*(3), 194–201.

Epstein, J. (2006). The calculus concept inventory. In *Proceedings of the National STEM Assessment Conference* (pp. 60–67).

Felleisen, M., Findler, R. B., Flatt, M., & Krishnamurthi, S. (2001). *How to design programs: An introduction to programming and computing*. Cambridge, MA: MIT Press.

Frost, R. (1993). *The Road Not Taken and other poems* (S. Applebaum, Ed.). Mineloa, NY: Dover Publications, Inc.

Goldman, K., Gross, P., Heeren, C., Herman, G., Kaczmarczyk, L., Loui, M. C., et al. (2008). Identifying important and difficult concepts in introductory computing courses using a Delphi process. In *SIGCSE '08: Proceedings of the 39th ACM technical symposium on computer science education* (pp. 256–260).

Hake, R. R. (1998). Interactive-engagement versus traditional methods: A six-thousand-student survey of mechanics test data for introductory physics courses. *American Journal of Physics*, *66*(1), 64-74.

Haladyna, T. M. (2004). *Developing and validating multiple-choice test items* (3rd ed.). Mahwah, NJ: Lawrence Erlbaum Associates, Inc.

Halloun, I. A., & Hestenes, D. (1985a). Common sense concepts about motion. *American Journal of Physics*, *53*(11), 1056–1065.

Halloun, I. A., & Hestenes, D. (1985b). The initial knowledge state of college physics students. *American Journal of Physics*, *53*(11), 1043–1055.

Hambleton, R. K., Swaminathan, H., & Rogers, H. J. (1991). *Fundamentals of item response theory*. Newbury Park, CA: Sage Publications.

Herman, G. L., Kaczmarczyk, L., Loui, M. C., & Zilles, C. (2008). Proof by incomplete enumeration and other logical misconceptions. In *ICER '08: Proceeding of the fourth international workshop on computing education research* (pp. 59–70).

Herman, G. L., Loui, M. C., & Zilles, C. (2010). Creating the digital logic concept inventory. In *SIGCSE '10: Proceedings of the 41st ACM technical symposium on computer science education.*

Hestenes, D. (1987). Toward a modeling theory of physics instruction. *American Journal of Physics*, *55*(5), 440-454.

Hestenes, D., & Wells, M. (1992). A mechanics baseline test. *The Physics Teacher*, *30*(3), 159.

Hestenes, D., Wells, M., & Swackhamer, G. (1992, March). Force concept inventory. *The Physics Teacher*, *30*, 141-158.

Horstmann, C. (2005). *Java concepts* (4th ed.). Hoboken, NJ: John Wiley and Sons.

Horstmann, C. (2006). *Big Java* (2nd ed.). Hoboken, NJ: John Wiley and Sons.

Hundhausen, C. D., Farley, S. F., & Brown, J. L. (2009). Can direct manipulation lower the barriers to computer programming and promote transfer of training?: An experimental study. *ACM Transactions on Computer-Human Interaction*, *16*(3), 1–40.

Kaczmarczyk, L. C., Petrick, E. R., East, J. P., & Herman, G. L. (2010). Identifying student misconceptions of programming. In *SIGCSE '10: Proceedings of the 41st ACM technical symposium on computer science education.*

Kane, M. T. (2006). Educational measurement. In R. L. Brennen (Ed.), (4th ed., pp. 17–64). Westport, CT: American Council on Education/Praeger Publishers.

Kelleher, C., & Pausch, R. (2005). Lowering the barriers to programming: A taxonomy of programming environments and languages for novice programmers. *ACM Computing Surveys*, *37*(2), 83–137.

Lewis, J., & Loftus, W. (2005). *Java software solutions (Java 5.0 version): Foundations of program design* (4th ed.). Boston, MA: Addison Wesley.

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

Lindquist, E. F. (Ed.). (1951). *Educational measurement.* Washington, D.C.: American Council on Education.

Lister, R., Adams, E. S., Fitzgerald, S., Fone, W., Hamer, J., Lindholm, M., et al. (2004). 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* (pp. 119–150).

Lukhele, R., Thissen, D., & Wainer, H. (1994, Fall). On the relative value of multiple-choice, constructed response, and examinee-selected items on two achievement tests. *Journal of Educational Measurement*, *31*(3), 234–250.

Malik, D. S. (2004). *C++ programming: From problem analysis to program design* (2nd ed.). Boston, MA: Thompson Course Technology.

Malik, D. S. (2006). *Java programming: From problem analysis to program design* (2nd ed.). Boston, MA: Thompson Course Technology.

Mathematical Sciences Education Board, & National Research Council. (1993). *Measuring up: Prototypes for mathematics assessment*. Washington, DC: National Academy Press.

McCracken, M., Almstrum, V., Diaz, D., Guzdial, M., Hagan, D., Kolikant, Y. B.-D., et al. (2001). A multi-national, multi-institutional study of assessment of programming skills of first-year CS students. *SIGCSE Bulletin*, *33*(4), 125–180.

Mercer, R. (2002). *Computing fundamentals with Java*. Wilsonville, OR: Franklin Beedle and Associates.

Miller, M. D., Linn, R. L., & Gronlund, N. E. (2009a). Constructing objective test items: Multiple-choice forms. In *Measurement and assessment in teaching* (10th ed., pp. 194–217). Upper Saddle River, NJ: Pearson Education.

Miller, M. D., Linn, R. L., & Gronlund, N. E. (2009b). Validity. In *Measurement and assessment in teaching* (10th ed., pp. 80–99). Upper Saddle River, NJ: Pearson Education.

Moss, P. A., Girard, B. J., & Haniford, L. C. (2006). Validity in Educational Assessment. *Review of Research in Education*, *30*(1), 109-162.

Neuendorf, K. A. (2002). *The content analysis guidebook*. Thousand Oaks, CA: Sage Publications.

Pintrich, P. R., & Schunk, D. H. (2002). *Motivation in education: Theory, research, and applications* (2nd ed.). Upper Saddle River, N.J.: Merrill.

Prather, E. E., Rudolph, A. L., Brissenden, G., & Schlingman, W. M. (2009). A national study assessing the teaching and learning of introductory astronomy. part i. the effect of interactive instruction. *American Journal of Physics*, *77*(4), 320–330.

Rogers, G. (n.d.). *Assessment: Could you please repeat the question?* [web page]. `http://www.abet.org`.

Sackman, H., Erikson, W. J., & Grant, E. E. (1968). Exploratory experimental studies

comparing online and offline programming performance. *Communications of the ACM*, *11*(1), 3–11.

Savitch, W. (2005a). *Java: An introduction to problem solving and programming* (4th edition ed.). Upper Saddle River, NJ: Prentice Hall.

Savitch, W. (2005b). *Problem solving with C++: The object of programming* (5th edition ed.). Boston, MA: Addison Wesley.

Schwartz, J. L., & Kenney, J. M. (2008). *Tasks and rubrics for balanced mathematics assessment in primary and elementary grades.* Thousand Oaks, CA: Corwin Press.

Shackelford, R. L. (1997). *Introduction to computing and algorithms.* Boston, MA: Addison Wesley.

Sime, M., Green, T., & Guest, D. (1976). Scope marking in computer conditionals: A psychological evaluation. *International Journal of Man-Machine Studies*, *9*, 107–118.

Smith, M. K., Wood, W. B., & Knight, J. K. (2008). The Genetics Concept Assessment: A New Concept Inventory for Gauging Student Understanding of Genetics. *CBE Life Science Education*, *7*(4), 422–430.

Snow, R. E. (1989). Aptitude-treatment interaction as a framework for research on individual differences in learning. In P. L. Ackerman, R. J. Sternberg, & R. Glaser (Eds.), *Learning and individual differences: Advances in theory and research* (pp. 13–59). New York, NY: W.H. Freeman.

Summet, J., Kumar, D., O'Hara, K., Walker, D., Ni, L., Blank, D., et al. (2009). Personalizing cs1 with robots. In *SIGCSE '09: Proceedings of the 40th ACM technical symposium on computer science education* (pp. 433–437).

Tew, A. E., & Guzdial, M. (2010). Developing a validated assessment of fundamental CS1 concepts. In *SIGCSE '10: Proceedings of the 41st ACM technical symposium on computer science education.*

Tew, A. E., McCracken, W. M., & Guzdial, M. (2005). Impact of alternative introductory courses on programming concept understanding. In *ICER '05: Proceedings of the 2005 international workshop on computing education research* (pp. 25–35).

The Joint Task Force on Computing Curricula (Ed.). (2001). Computing curricula 2001. *Journal on Educational Resources in Computing*, *1*(3es), 1–240.

Weinberg, G. M. (1971). *The psychology of computer programming.* New York, NY: Van Nostrand Reinhold Company, Inc.

Whitfort, T. (n.d.). *Pseudo code guide* [web page]. `http://ironbark.bendigo.latrobe.edu.au/subjects/PE/2005s1/other_resources/pseudocode_guide.html`.

Wu, C. T. (2006). *Intro to object oriented programming using Java* (4th ed.). Boston, MA: McGraw Hill.

Zelle, J. M. (2004). *Python programming: An introduction to computer science.* Wilsonville, OR: Franklin Beedle.