PREDICTING STUDENT PERFORMANCE IN INTRODUCTORY COMPUTER PROGRAMMING COURSES

by

William E. J. Doane

A Dissertation

Submitted to the University at Albany, State University of New York

in Partial Fulfillment of

the Requirements for the Degree of

Doctor of Philosophy

College of Computing & Information

Department of Informatics

May 2008

Predicting Student Performance in Introductory Computer Programming Courses

by

William E. J. Doane

Abstract

For decades, computer science education researchers have sought to improve computing education by refining curricula, instructional methods, and choice of first language. Central to the task of improving computer science education is the identification of students in need of assistance, ideally as early in their academic career as possible. Until recently, no known assessment instrument offered a good predictor of student performance in introductory computer programming courses. Such an instrument, should it be created, would allow educators to identify students who would be likely to have difficulty learning to program. It would also allow instructors to design instruction intended to support those students, and to allocate instructional resources more appropriately. In 2006, researchers in the United Kingdom identified an assessment instrument that shows promise as a predictor of students' final grades in introductory computer programming courses. In that same year, researchers in Massachusetts found that the commercially available logic puzzle *MasterMind®* also showed promise as a predictor of in-class programming test scores. What connects these techniques? What makes them more successful than past assessment instruments designed to test programming potential?

In this study, novice programmers at the undergraduate and high school levels completed a modified version of the paper-based assessment instrument designed by U.K. researchers. Students also were asked to complete web-based tasks based on *MasterMind®* and Sudoku.

The purpose of this study was to collect information about novice computer programming students and to use that information effectively to predict their final numeric course grade in introductory computer programming courses. In order to extract all of the most relevant information from the initial collected data effectively, both model-based and algorithmic prediction methods were used in the predictive analyses. Regression trees were used, in addition to model-based multiple linear regression methods, to derive both generalizable and interpretable predictive results, given the available datasets.

Table of Contents

Figures

Tables

Chapter 1. Introduction

Some ten years ago, Allen Tucker proclaimed that there was a crisis in computer science education (Tucker, 1996a). He identified outdated curricula, inadequate teaching methods and faculty development, and a fragmented discipline as core challenges to the future of computer science education. Others have more recently identified lack of problem-solving skills, high dropout rates, and decreased student enrollment as core concerns (Ma, Ferguson, Roper, & Wood, 2007; McBride, 2007). For decades, computer science education researchers have sought to improve computing education by refining curricula, instructional methods, and choice of first language (Tucker et al., 2003). Others have sought indicators of students' computer programming potential with little success (Dehnadi, 2006; Lorenzen & Chang, 2006). In particular, questions regarding novice programmers' prior experiences, successes, and failures have been a focus of much research (Bayman & Mayer, 1983; Du Boulay, 1989; Lister et al., 2004; Ma et al., 2007; Mayer, 1981; Soloway & Spohrer, 1989).

Until recently, no known assessment instrument offered a good predictor of student performance in introductory computer programming courses (Lorenzen & Chang, 2006). Such an instrument, should it be created, would allow educators to identify students who would be likely to have difficulty learning to program. It would also allow instructors to design instruction intended to support those students, and to allocate instructional resources more appropriately. In 2006, researchers in the United Kingdom designed an instrument (referred to in this study as *Dehnadi's assessment instrument* or simply *Dehnadi's instrument*) that shows promise as a predictor of student performance in introductory computer programming courses (Dehnadi, 2006; Dehnadi & Bornat, 2006). Also in 2006, researchers in Massachusetts found that the commercially available logic puzzle *Mastermind®* also showed promise as an indicator of students' in-class programming test scores (Lorenzen & Chang, 2006). What connects these techniques? What makes them more successful than previous tests designed to test programming potential?

This study posits that the logical, step-wise reasoning required by these assessment instruments reflects habits of mind that undergird the reasoning of students who perform well in introductory programming courses. Learners' mental models regarding the lawfulness and consistency of the task of interest influence their ability to learn to program a computer using languages in which assignment follows the rule: *the value of the left-hand-side becomes the value of the right-hand-side and the right-hand-side keeps its value*. Java and JavaScript are examples of such languages. (Java and Java-style languages are the de facto standard for modern introductory programming courses. Java is the chosen language for the high school Advanced Placement course offered in the United States.)

The goal of this research was to identify assessment instruments that could be used as predictors of student performance in introductory programming courses and to use the information provided by those instruments optimally. A table of variables of interest is included in Appendix A. Early-course Variables of Interest including their descriptions and source.

Two of the assessment instruments included in this study were drawn from recent research on computer programming aptitude: Dehnadi's assessment instrument and

Mastermind®. Dehnadi's instrument was designed to reveal learners' mental models regarding *assignment of values to variables* and to evaluate the *consistency* and *viability* of those mental models (Dehnadi, 2006). Consistency of mental model was believed to be more important than viability of the model. That is, if the learner believes there is one and only one rule that applies, and if they apply that rule consistently, then the initial correctness of the rule appears to be unrelated to the learner's performance in introductory programming (Dehnadi, 2006). *Mastermind®* purportedly revealed programming aptitude by requiring players to apply "...analytic skills and logical deductive powers..." in order to guess the correct solution to a puzzle (Lorenzen & Chang, 2006). Both Dehnadi's assessment instrument and *Mastermind®* relied on the learner's ability to apply a lawful process in order to find a solution. Indeed, if the learner believes the process to be arbitrary, then a correct solution is the result of pure chance. Based on this, an additional instrument was included in this study: Sudoku.

Sudoku is a number puzzle game where the player uses a set of initial constraints (filled-in numbers) and deductive skills to discover the placement of all other numbers in the puzzle (for a complete description, see Appendix G. Sudoku Game). Sudoku, like Dehnadi's assessment instrument and *Mastermind®*, requires the consistent application of lawful rules in order to arrive at a solution.

If, indeed, such tasks can be used to predict final performance, then computer science educators can use this information to redesign introductory courses and better serve students and the discipline. This study sought to make optimal use of information collected early in the course in order to predict students' final course grade. To the degree this is possible, interventions may be targeted at those students identified as likely to have low final grades.

State of Computer Science Education

Computing has become a vital facet of almost every area of modern human endeavor. Computing technologies have transformed how we engage with the world around us (Kling, 1996). Computing-related jobs are growing at a staggering rate. The United States Department of Labor projects that between 2004 and 2014, "...computer systems design and related services will grow by 39.5 percent and add almost one-fourth of all new jobs in professional, scientific, and technical services" (United States Department of Labor, 2003). Others note that, in the future, the larger information technology field will represent nearly one million new jobs (Blake, 2006). Indeed, "computer occupations account for 5 out of the 20 fastest growing occupations in the economy", as shown in Figure 1, while healthcare accounts for 12 of the remaining occupations (United States Department of Labor, 2003).

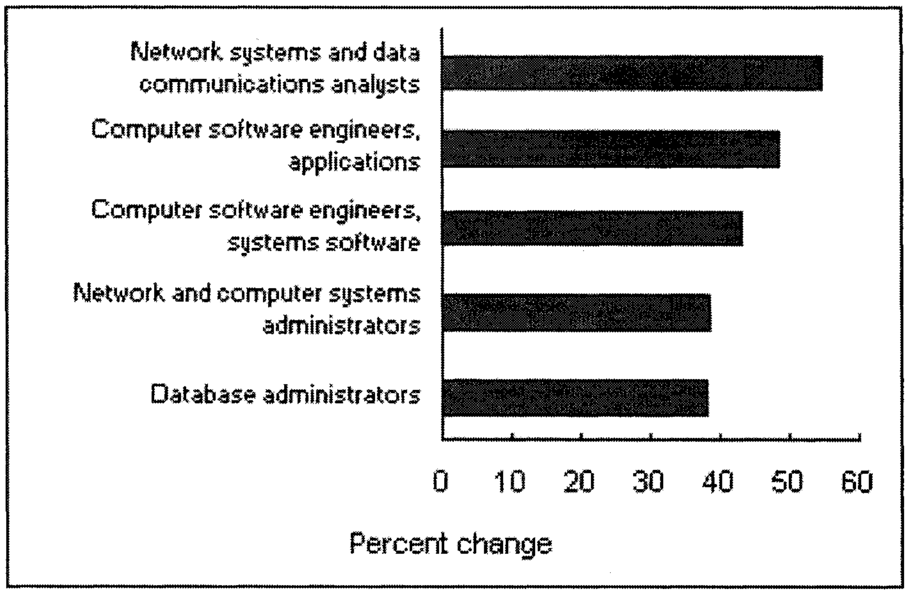


Figure 1: Projected percent change in employment, 2004-2014 (Adapted from: United States Department of Labor, 2003)

Despite the projected increase in computing jobs, enrollment in undergraduate computer science programs decreased by nearly 60 percent between 2000 and 2004 (Vegso, 2005), as shown in Figure 2, and the downward trend continued in 2005 (Blake, 2006). Additionally, the rate of graduating women and minority students from computer science programs slowed during the same period (Levy, 2007). Clearly, there is a widening gap between the demand for highly skilled computer professionals and our ability to entice students into the field and to engage them once they are there.

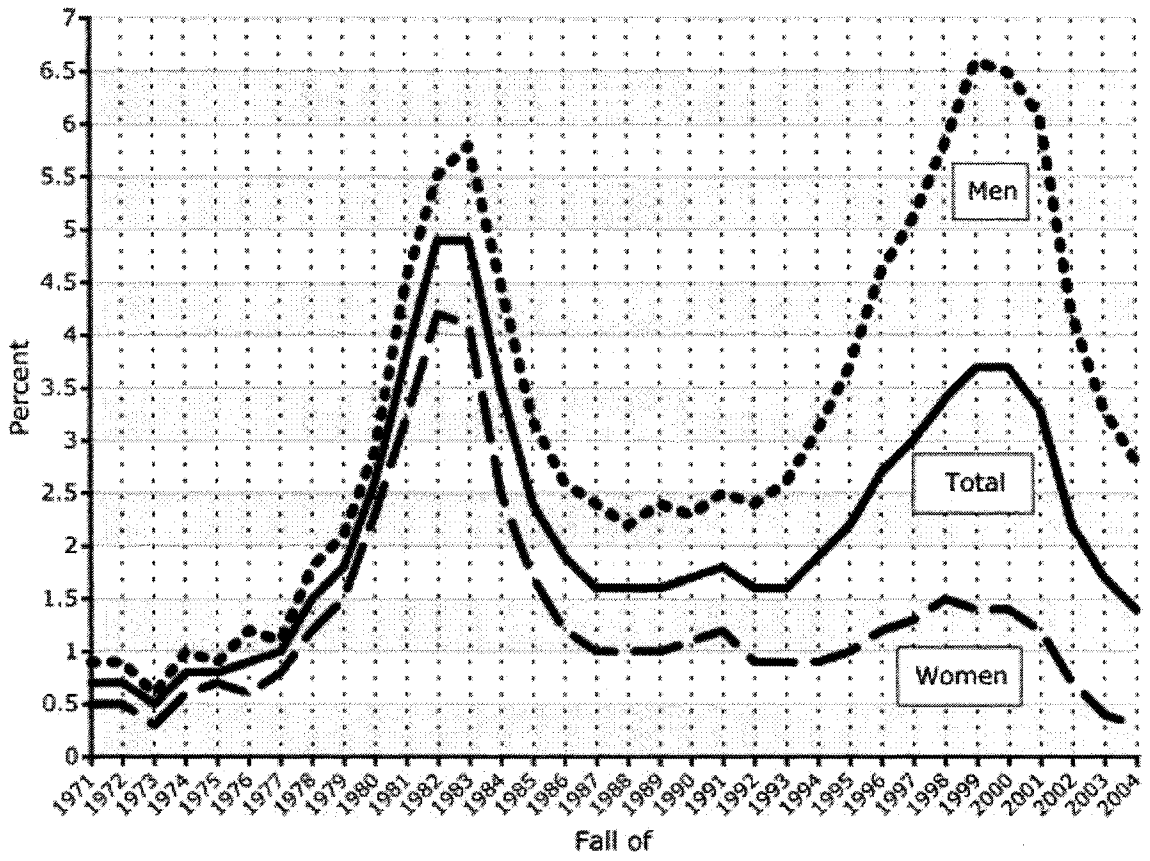


Figure 2: Percentage of freshmen listing computer science as a probable major (Vegso, 2005)

Computer Science, Teaching it, and Understanding it

The first computer science (CS) departments began to emerge in the 1960s (Tucker, 1996b), only 24 years after Alan Turing first proposed what later became known as the Turing Machine, an abstract model of a computing machine capable of calculating any computable number (Turing, 1936). As digital computers were designed and built, the introduction of formal, professional training in computer science began to raise questions regarding the content, order, and delivery of the new CS curriculum (Tucker, 1996b). These issues remain the focus of computer science education (CSEd) today.

Broadly defined, CSEd is the teaching and learning of CS-related topics. CSEd embodies teaching methods, learning theories, and curriculum development with regard to CS (Fincher & Petre, 2004).

The desire to understand CSEd and the successes and failure of computing education gives rise to the discipline of computer science education research (CSEdR). Researchers of CSEd come from a broad range of disciplines including "...at least... education, psychology, computer science, technology and engineering" and consider questions relating to:

- "student understanding,
- animations/visualization/simulation systems,
- teaching methods,
- assessment,
- educational technology,
- the transfer of professional practice into the classroom,
- the incorporation of new development[s] and new technologies into the classroom,
- transferring to remote teaching ('e-learning'),
- recruitment and retention of students, and...
- the construction of the discipline itself" (Fincher & Petre, 2004).

The abilities to conceptualize and predict the functions of computer systems and to program computers to attain desired results are critical to success in computer science (Stephenson, Gal-Ezer, Haberman, & Verno, 2005). Unless students can conceptualize the inner workings of the computer well, their expectations of program execution will often be incorrect (Du Boulay, O'Shea, & Monk, 1989). Accordingly, one active area of CSEdR has been the learning of programming languages, particularly by novice programmers: those students who have not previously learned another programming language (see for example studies by Ben-Bassat Levy, Ben-Ari, & Uronen, 2003; Brusilovsky, Kouchnirenko, Miller, & Tomek, 1994; Clancy, 2004; Dehnadi, 2006; Duke, Salzman, Burmeister, Poon, & Murray, 2000; Fay & Mayer, 1988; Fincher, 1999; Guzdial, 2004; Karsten & Kaparthi, 1998; Mayer, 1988a, 1988b; Perkins, Schwartz, & Simmons, 1988; Robins, Rountree, & Rountree, 2003; Weinberg, 1988; Yuen, 2006).

Programming Computers

"Computer programmers write, test, and maintain the detailed instructions, called programs, that computers must follow to perform their functions. Programmers also conceive, design, and test logical structures for solving problems by computer" and "computer programs tell the computer what to do—which information to identify and access, how to process it, and what equipment to use" (United States Department of Labor, 2007). As such, the task of programming a computer is central to the successful operation of a computer system. A poorly designed or poorly coded computer program can result in unstable system performance and computer failures. Programming is one critical aspect of computing. "Clearly programming is part of the standard practices of the discipline and every computing major should achieve competence in it" (Denning et al., 1989).

Much modern computer programming is done using so-called high level languages. High level languages provide layers of abstraction between the computer's hardware and the programmer. The programmer can instruct the computer to add two numbers, without regard to the underlying mechanisms that the computer employs to perform the addition. However, as programming becomes increasingly removed from the underlying hardware, novice programmers are called upon to learn not only the programming language for providing instructions to the computer, but also to intuit one or more models of the underlying physical and computational processes involved in the computer's operation. These "deep conceptual hierarchies... [have] no precedent in our history" (Dijkstra, 1988) and pose significant challenges to learners. That is, the instructions given to the computer using a high level language are so far removed from the underlying hardware implementation that programmers must understand and model both theoretical and physical systems to which they have no direct access. This introduces a complexity to the task unprecedented in human history.

Mental Models

In order to make sense of complexity, people construct mental models that represent critical aspects of a phenomenon of interest (Johnson-Laird, 1983). When learners begin learning a programming language, they may possess mental models of how computers operate and how computer programs function based on their prior experience (Day & Kovacs, 1996). These models may be *viable* (i.e., reasonably model critical aspects of the phenomenon) or *non-viable* (i.e., lack or misrepresent critical aspects) (Johnson-Laird, 1983). These models are used to interpret the observable operation of the computer and to intuit the unobservable aspects of a computer's operations.

Mental models are a critical aid to learning and lack of viable mental models can undermine the learning process (Day & Kovacs, 1996). However, fostering viable mental models requires that "[o]ne must take care... to make clear the correspondence between the model presented and the modeled process. For example, the dynamic process of program execution is difficult to convey with a static description" (Clancy, 2004, p. 95). In order to understand learners' current mental models, we must gain insight into the organization and content of those models.

Researchers in the United Kingdom believe they have developed an assessment instrument (referred to in this study as *Dehnadi's assessment instrument*) that reveals learners' mental models concerning the assignment of values to variables (Dehnadi, 2006; Dehnadi & Bornat, 2006). If the instrument can be validated, it promises to offer insight into who is likely to succeed in introductory programming courses, as indicated by measures of student performance. Assuming that excellent performance in programming courses is an indicator of programming potential, researchers have long sought such a predictor of programming potential (see papers by Bayman & Mayer, 1983; Bonar & Soloway, 1989; Brusilovsky et al., 1994; Clancy, 2004; Cunniff, Taylor, & Black, 1989; Du Boulay, 1989; Du Boulay et al., 1989; Kuittinen & Sajaniemi, 2004; Mayer, 1981; Mayer, Dyck, & Vilberg, 1986; Pea, 1986; Perkins, Hancock, Hobbs, Martin, & Simmons, 1989; Robins et al., 2003; Sajaniemi & Kuittinen, 2005). In particular, the role of variables has been of concern, since so many novice programmers seem to have difficulty understanding the nuance of variables (Kuittinen & Sajaniemi, 2004; Sajaniemi & Kuittinen, 2005). Finding correlates to student performance in introductory computer programming courses has been so elusive, some researchers have even looked to logic games as a possible predictor—with significant success (Lorenzen & Chang, 2006). These studies are discussed in further detail, below.

Introduction to the Study

In this study, novice programmers at the undergraduate and high school levels completed a modified version of Dehnadi's assessment instrument, a paper-based computer programming instrument designed by U.K. researchers. Additionally, students were asked to complete web-based tasks based on *Mastermind®* and Sudoku. The intention was to use the information from these instruments in order to predict students' final course grade. The response rate for the online games prevented data from those instruments from being used in the data analysis conducted.

Participants completed a modified version of Dehnadi's instrument during the first week of introductory computer programming classes. This instrument provided **demographic information** (age, gender, and prior experience) on participants, responses to **individual items**, and data concerning the consistency of **learners' mental models** (based on Dehnadi's scoring methods). Following the administration of Dehnadi's instrument and prior to instruction regarding the assignment of value to variables, participants were asked to complete the online games. An automated system collected data regarding **participant actions** during game play including **errors made** and **time-to-complete tasks.** At the end of term, after final grades have been processed, the **final grades** of participants were collected and served as the criterion variable for this study.

Participants were solicited from university and community college computer science students. In addition, some participants will be students enrolled in Johns

Hopkins' summer program for gifted youth. These institutions have been selected largely for reasons of practicality—in particular their proximity to the researcher's home institution in order to facilitate administration and oversight of assessment administration.

Research Questions

This study, in part, emulates a study conducted recently in the United Kingdom concerning the mental models of novice programming students (Dehnadi, 2006). The instrument was modified to include several questions relating to logic and discrete mathematics as well as several qualitative questions regarding participants' reactions to the instrument. Additionally, two logic-based games were introduced in an effort to identify which, if any, predict final course grade in introductory computer programming courses. The questions addressed by this study were:

- What variables or combination of variables collected at the start of term serve best as predictors of students' final grades at the end of term?
- Following Dehnadi, is the measure of consistency of students' mental models regarding assignment of values to variables at the outset of the course positively correlated with students' final grades at the end of the course? If so, how strong is that relationship?
- Are there additional predictor variables that tend to moderate or condition the relationship(s) between the main 'consistency' measures and the outcome measure? If so, what is the nature and extent of the moderator effects?

- What are the similarities and differences among the predictability results across the three main groups: university students, community college students, and (high performing) high school students?

Significance

This study contributes to our understanding of the relationship(s) between student success in introductory computer programming courses (as indicated by final course grade) and students' initial mental models of the processes underlying assignment of values to variables and ability to reason logically (as indicated by questions added to Dehnadi's instrument and the online tasks).

The authors of the U.K. studies suggested that the ability to distinguish between those likely to succeed in a programming course and those likely to fail offers a powerful administrative tool that could be used to determine which students are allowed to continue their study of computer programming, and which are lost causes (Dehnadi & Bornat, 2006). While that idea derives mostly from considerations of resource allocation and admission to a program of study, it precluded the possibility that identification of students who are likely to have a notable difficulty in such courses might be helped at the outset to attain the mental model skills that may serve to make them successful in programming courses. Several other possibilities might be considered: students could use the instrument as a self-diagnostic tool to gauge their skill for computer programming; instructors could use the information provided by such an instrument to develop remedial lessons that would aid those identified as likely to fail (although the effectiveness of such remediation remains an open research question).

Additional question arise regardless of the findings pertaining to the role of measures of consistency of mental models of *assignment of values to variables* for prediction of final grade in Java-style programming courses. What mental model measures might be effective as predictors of success in courses based on other programming languages, such as LISP or Prolog? How might students' initial mental models inform common student misconceptions and programming errors? What additional variables might moderate or condition the predictability of outcomes in such different contexts?

In short, this study broadens a line of research that could inform student selection of majors as well as instructional planning, curriculum design, and assessment in computer programming courses. This study has the potential to benefit students, educators, and the field of computer science education by identifying diagnostic tools and knowledge with potentially wide-ranging application and effect.

Chapter 2. Related Literature

This study draws upon work in cognitive analysis, mental representations, and the psychology of computer programming. Broadly, it assumes that expert and novice computer programmers possess mental models of computational processes and that these models differ between the two groups (possibly because of training or experience). Further, it assumes, in accord with accepted mental model theory, that such models are mutable, albeit unknowable, and that at least some novices may, through experience and training, modify their mental models to resemble more closely those of expert programmers (Johnson-Laird, 1989).

The proposed study contributes to work in computer science education (CSEd) and computer science education research (CSEdR). CSEd seeks to produce competent computer science (CS) professionals, including computer programmers. CSEd is concerned with the selection of curriculum, delivery of instruction, and assessment of learning in CS. CSEdR, then, is the systematic study of the effects of choices made in CSEd and the exploration of alternative curricula, instructional methods, and assessment techniques (Fincher & Petre, 2004). CSEdR is a young discipline, necessarily lagging behind CS and CSEd, and "...is too new, and there are too few people doing work in this field. We are still in the stage of the field of identifying potential answers to key questions - indeed, even figuring out what the key questions are! " (Guzdial, 2004, p. 128). "Historically, computing technology has been far ahead of computer science pedagogy. Each new programming paradigm - procedural, functional, declarative, and object-oriented programming - has its own computational model, usage idioms, and so on" (Clancy, 2004, p. 97) and both educators and learners are challenged by these ever shifting sands.

Mental Models

Mental models are internal representations of a believed state of the world and relationships among elements in the world (Johnson-Laird, 1989). Mental models are, by their nature, unknowable and inexpressible (Johnson-Laird, 1983). "Our models profoundly influence how and why we act, teach, and learn in the ways we do " (Henderson & Tallman, 2005, p. 18). Mental models are based on an information processing model of mind (Johnson-Laird, 1983). Accordingly, they are subject to Shannon-style constraints: imperfection, correction, and transmission (Shannon, 2001). When someone attempts to express a given mental model, the representation of the model is a necessarily imperfect representation. Mental models imperfectly represent a fact or relationship about the world. They can be expressed, introducing error into the representation. Those expressions can then be interpreted by another individual and converted into another mental model, necessarily an imperfect model of the representation. In short, the conveyance of mental models from one individual to another is an error prone, yet absolutely necessary element of learning (Johnson-Laird, 1988).

People construct mental models that represent critical aspects of a phenomenon of interest in order to make sense of complexity (Johnson-Laird, 1983). These models incorporate what the individual perceives as salient aspects of the phenomenon and omit non-salient aspects (P. A. Smith & Webb, 1995). When learning a programming language, learners may hold mental models of how computers operate and how computer programs function based on their prior experience (Day & Kovacs, 1996). These models may be viable (i.e., reasonably model critical aspects of the phenomenon) or non-viable (i.e., lack or misrepresent critical aspects) (Johnson-Laird, 1983; Ma et al., 2007). These models are used to interpret the observable operation of the computer and to intuit the unobservable aspects of a computer's operations.

Since programmers cannot directly observe the systems they affect (typically, one does not directly observe the off and on switches present in computers), possessing viable mental models of the underlying processes is critical to success in programming. Nevertheless, one-third of the students completing programming courses do not hold viable mental models of assignment of values to variables (Ma et al., 2007).

While there has been significant research on eliciting mental models in general (Chittleborough, Treagust, Mamiala, & Mocerino, 2005; Ehrlich, 1996; Gagné & Glaser, 1987; Garnham, 1987; Henderson & Tallman, 2005; Johnson-Laird, 2005), little work has been done in CSEdR to determine what mental models novice programmers hold (Ma et al., 2007).

Novice Programmers

A recent study estimated that the average failure rate in introductory programming courses world-wide is 33% (Bennedsen & Caspersen, 2007). That study found no correlation between the way in which the course was taught, the focus of the course, the programming language used, and whether students passed or failed the course.

Early proponents of teaching computer programming broadly and at a young age touted the benefits that such training would have for learners: "In teaching the computer how to think, children embark on an exploration about how they themselves think" (Papert, 1980, p. 19). Research has shown that "[n]o matter what the age of the students, programming is hard to learn. Whether students attempt to learn to program at a young age or at the age of young adults, the tasks and difficulties remain similar" (Guzdial, 2004, p. 128). Reflecting this, attempts to quantify the benefits to learners have resulted in "...mounting data point[ing] to problems in students' learning..." (Mayer, 1988a, p. 2) and "...studies of the relationship between higher-order thinking skills and programming have never shown any significant correlation" (Guzdial, 2004, p. 131).

Understanding what difficulties novice programmers have with learning to program and what misconceptions they have regarding program execution are valuable to improving computer programming instruction and learning (Mayer, 1981; Mayer et al., 1986; Soloway & Spohrer, 1989). Misconceptions may indicate non-viable mental models held by novice programmers (P. A. Smith & Webb, 1995). "...if the programmer has a faulty mental model of how the computer works then she will have great difficulties in creating correct programs to solve her problems" (P. A. Smith & Webb, 1995, p. 2). Thus, "...one of the first learning tasks a novice programmer must undertake is to obtain an adequate mental model" (P. A. Smith & Webb, 1995, p. 2).

Research into the experiences of novice programmers has identified sources of errors made by novice programmers (Clancy, 2004). These include:

- linguistic transfer: English words take on special meaning in computer programming,

- algebraic notation: relying on their understanding of algebra to interpret computer code, and
- Over- or under-generalization: incorporating too much or too little detail from examples.

Nevertheless, "...cataloging and analyzing misconceptions will not be sufficient to improve students' misunderstanding" (Ben-Ari, 1998), so we must seek out the underlying cause of the misconceptions. "Students' fragile knowledge of programming [is attributable] in considerable part to a lack of a mental model of the computer that helps learners to encode and consolidate their knowledge of programming" (Perkins et al., 1988, p. 162). Thus, mental models provide one avenue of research to explore and explain the root causes of novice misconceptions. This is reflected in the literature on science education, where "there is a considerable literature on how misconceptions and inappropriate attitudes complicate learning" (Clancy, 2004, p. 85).

The confusion caused by the similarity between programming statements and algebraic notation (Clancy, 2004) is worth additional comment. In standard algebra, with which most novice programmers are familiar, the pairs of statements given in Table 1 are equivalent. Yet, in Java-style languages, each statement is quite different. Indeed, '10 = x' is a malformed statement and would prevent a program from compiling successfully, while 'a = b' and 'b = a' have very different effects, given the LEFT-HAND-SIDE BECOMES THE VALUE OF RIGHT-HAND-SIDE method of assignment of values to variables.

Table 1. Sample algebraic equivalences; equivalent in algebra, but not in computer programming

x = 10
 	
10 = x
	a = b
	
	b = a

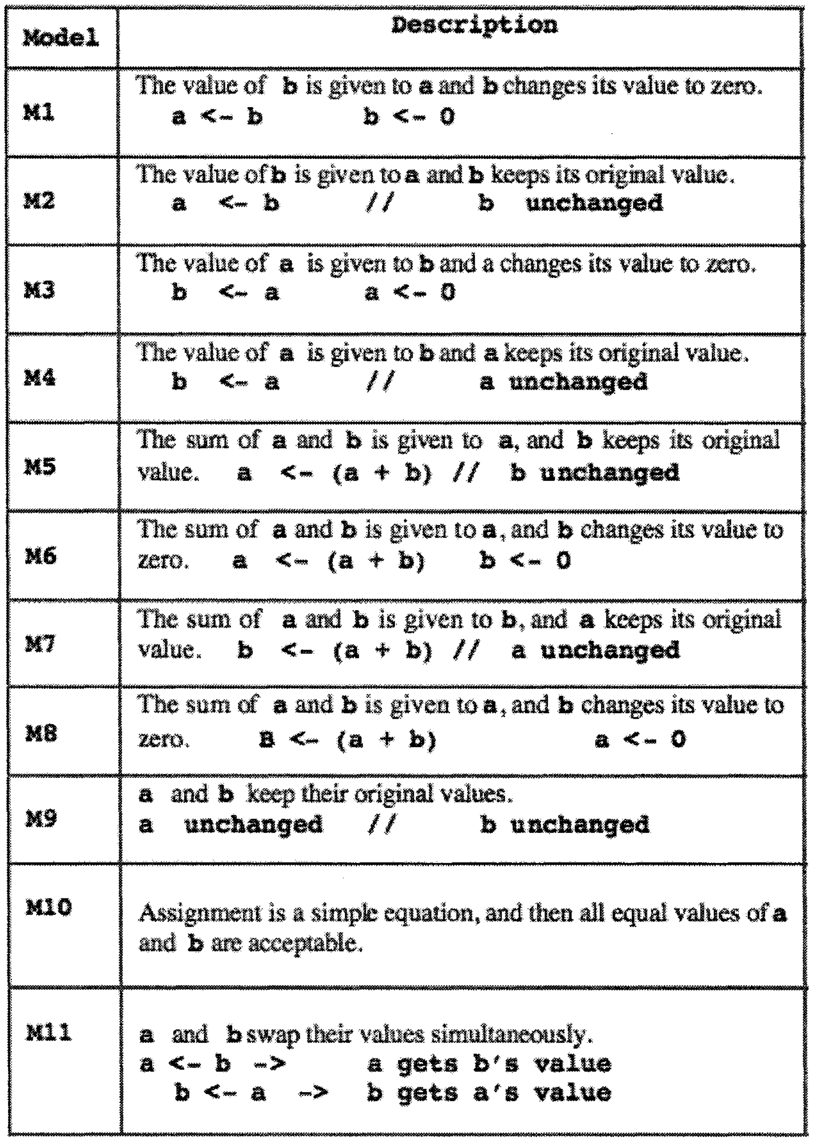
If learners hold an algebraic model of assignment, then programming errors are inevitable. Understanding learners' mental models is a necessary first step to improving teaching and learning of computer programming, since "[learners'] misconceptions spring from a learner's efforts to link knowledge. Remedies must focus on that linkage to foster more coherent understanding" (Clancy, 2004). "What is at issue is ... whether there is any pedagogical advantage in providing people with models of tasks they are trying to learn" (Johnson-Laird, 1989, p. 485).

Dehnadi's Assessment Instrument

The paper-based instrument adopted for this study attempt to make manifest students' mental models and to predict, based on the consistency of those models, who is likely to do well versus who is likely to do poorly in an introductory computer programming course. It was first presented in the United Kingdom in 2006 and administered to 60 computer programming students at the School of Computing, Middlesex University. (Dehnadi, 2006; Dehnadi & Bornat, 2006)[1]. Researchers sought to identify which incoming computer programming students were likely to have difficulty

[1] Dehnadi has graciously given permission for his instrument to be used in this study.

with the introductory programming course and which were likely to pass relatively easily. To answer this question, they devised a short test that focused on eliciting students' mental model(s) regarding the assignment of values to variables (included as Appendix C. Modified Dehnadi's Assessment). Each possible response was analyzed and classified as representing one of 11 mental models of assignment, as summarized in Figure 3. The order of the models represented in the response set on the instrument was mixed, so that a student blindly accepting all of the first responses, for example, would not appear to have a consistent mental model of assignment.



Model	Description
M1	The value of **b** is given to **a** and **b** changes its value to zero. `a <- b        b <- 0`
M2	The value of **b** is given to **a** and **b** keeps its original value. `a  <- b       //      b  unchanged`
M3	The value of **a** is given to **b** and a changes its value to zero. `b  <- a       a <- 0`
M4	The value of **a** is given to **b** and **a** keeps its original value. `b  <- a       //      a unchanged`
M5	The sum of **a** and **b** is given to **a**, and **b** keeps its original value. `a  <- (a + b) //  b unchanged`
M6	The sum of **a** and **b** is given to **a**, and **b** changes its value to zero. `a  <- (a + b)    b <- 0`
M7	The sum of **a** and **b** is given to **b**, and **a** keeps its original value. `b  <- (a + b) //  a unchanged`
M8	The sum of **a** and **b** is given to **a**, and **b** changes its value to zero. `B <- (a + b)          a <- 0`
M9	**a** and **b** keep their original values. `a  unchanged   //      b unchanged`
M10	Assignment is a simple equation, and then all equal values of **a** and **b** are acceptable.
M11	**a** and **b** swap their values simultaneously. `a <- b  ->      a gets b's value` `b <- a  ->  b gets a's value`

Figure 3: Mental model identifiers and their associated descriptions (Dehnadi, 2006)

In order to determine what mental models students held regarding the assignment of values to variables and to assess the consistency of those models, the U.K. study assessed consistency at four levels (see Appendix D. Dehnadi's Scoring Form), where each "higher" level represents a more general model (simple assignment versus directional assignment, e.g.):

- C0: This is the most detailed mental model. At this level, the student's mental model reflected assignment or addition-assignment (where the Left Hand Side [LHS] takes on the value of itself PLUS the Right Hand Side [RHS]) of values to variables, either to the left or to the right, and either always losing the RHS value or always keeping it.
- C1: At this level, the student's mental model reflected assignment of values to variables, either to the left or to the right, but sometimes the RHS loses its value and sometimes keeps it.
- C2: At this level, the student's mental model reflected only assignment or addition-assignment.
- C3: At this level, the student's mental model reflected only that there was an operation taking place, but as not specific as to how that operation took place.

The hierarchy of the models is depicted in Figure 4. For reference, the "correct" mental model of assignment of values to variables in Java-style languages is indicated by the inverted node, namely assign-to-left, with the right-hand-side keeping its value. Java and Java-style languages are the de facto standard for modern introductory programming courses and Java is the chosen language for the high school Advanced Placement course currently offered in the United States.

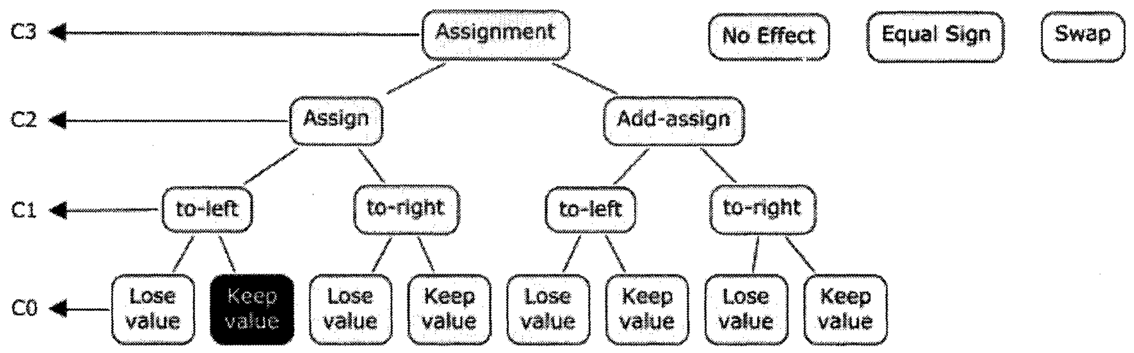


Figure 4: Depiction of levels of consistency in student mental models

Students in the U.K. study were said to have a *consistent mental model* of assignment at a given level if and only if at least eight of the responses to the instrument reflected the same mental model. Otherwise, students were said to have an *inconsistent mental model* of assignment at that level. Trivially, a student consistent at level C0 is also consistent at levels C1, C2, and C3.

They discovered that those students who espoused a consistent mental model (regardless of whether or not the model was the correct model with respect to Java-style assignment [M2]) were significantly more likely to pass the course than those who espoused inconsistent mental models (Dehnadi, 2006). The distribution of final scores (letter-based and numeric) is presented in Figure 5 and Figure 6, respectively. Each figure clearly shows the separation between those with a consistent mental model and those without, while the variance is more pronounced in Figure 6.

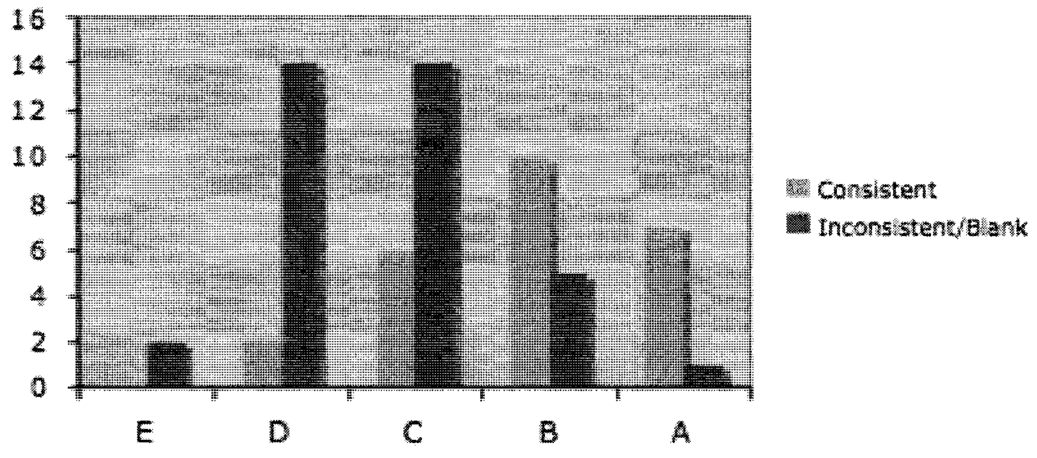


Figure 5: Number of students receiving the indicated letter grade

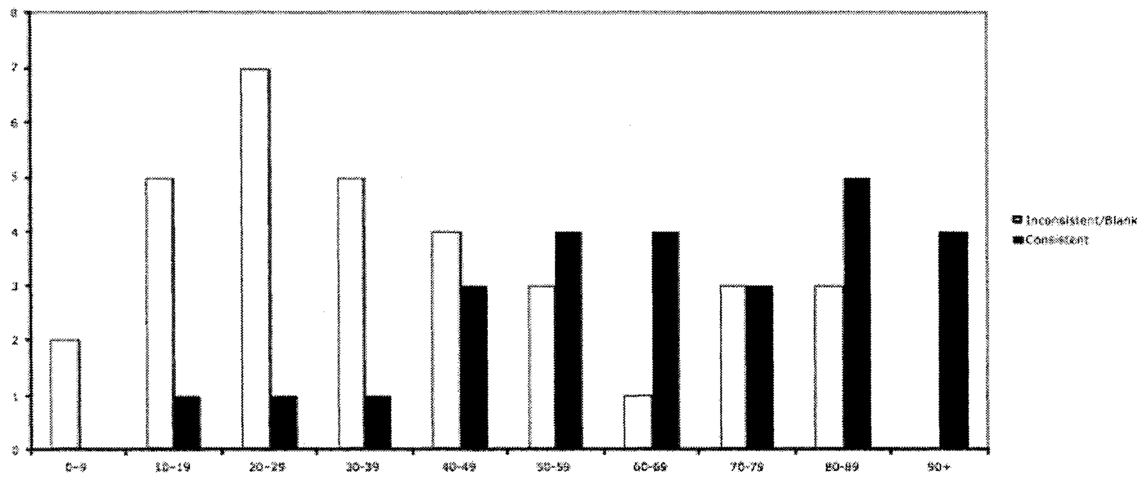


Figure 6: Number of students receiving the indicated numeric grade

Students were said to have a *consistent mental model* of assignment if and only if at least eight of the responses reflected the same mental model. Otherwise, students were said to have an *inconsistent mental model* of assignment. Some students did not respond and were included in the "inconsistent" group (Dehnadi, 2006).

While much work remains to validate the findings of the U.K. group, initial response to the results of the test has been positive (Ma et al., 2007). Other research teams are currently working to replicate the study in Australia, Canada, Ireland, Germany, New Zealand, and Mozambique (Dehnadi, 2007).

Mastermind®

The commercially available logic game, *Mastermind®*, involves two players: the code-maker (CM) and the code-breaker (CB). The CM selects four marbles from a set of eight possible colors and orders them. The CB then has twelve guesses to find the correct colors and ordering. After each guess by the CB, the CM indicates how many of the marbles guessed are both the right color and the right position by placing a dark pin to the side and how many are the correct colors, but not the correct position, by placing a light colored pin. For example, if the solution was red-red-blue-black, and the guess was red-white-black-blue, them the CM would place one dark pin (for the correct "red" marble) and two light pins (for the misordered black-blue marbles). Given the information provided by previous guesses and the responses from the CM, the CB attempts to deduce the solution. This process engages the CB in an iterative process of seeking for the solution, which he or she can do if and only if he or she believes that the process is lawful.

Lorenzen and Chang used this game as a basis for a computer programming aptitude test (2006). They believe that the analytical and logical skills required by the game parallel those used by programmers. If this is the case, then other games that require similar analytical, logical reasoning may also serve as predictors of student performance.

Improving Programming Education

Researchers have considered a wide range of predictors for success in introductory programming courses including choice of first programming language (Bayman & Mayer, 1983; Cunniff et al., 1989; Duke et al., 2000; Xinogalos, Satratzemi, & Dagdilelis, 2006), choice of development environment (Guzdial, 2004; Xinogalos et al., 2006), gender (Dean, 2007; Linn, 1995; Margolis & Fisher, 2003; McKenna, 2000), age, prior programming experience, math experience, attitude, and learning style (Bonar & Soloway, 1989; Brusilovsky et al., 1994; Clancy, 2004; Du Boulay, 1989; Gray, Goldberg, & Byrnes, 1993; Kahney, 1989; Kay, ; Ma et al., 2007; Mayer, 1981; Mayer et al., 1986; Milne & Rowe, 2002; Pea, 1986; Perkins et al., 1989; Perkins et al., 1988; D. C. Smith, Cypher, & Tesler, 2000; P. A. Smith & Webb, 1995; Wu, Dale, & Bethel, 1998). Recently, one study considered the use of the children's board game *Mastermind®* as a possible predictor of in-class programming test scores—with greater success than many of the above studies (Lorenzen & Chang, 2006).

While much research has been conducted to locate accurate predictors of student performance in introductory computer programming courses, there has been little success (Dehnadi, 2006; Lorenzen & Chang, 2006). An accurate predictor of student performance in introductory programming courses would allow educators to develop targeted instruction for those students likely to have difficulty in the course and would provide a rational basis for instructional design.

Chapter 3. Methods

The purpose of this study was to collect information about novice computer programming students and to use that information effectively to predict their final numeric course grade in introductory computer programming courses. Ultimately, this work may be used to break those predictions, by identifying students likely to have difficulty in introductory programming courses and allowing computer science educators to craft instructional interventions to aid those students.

This study considered three groups of students studying either JavaScript or Java, similar but distinct programming languages:

1. Johns Hopkins' Center for Talented Youth gifted and talented high school students learning JavaScript;
2. university undergraduates learning Java; and
3. community college students learning Java.

Each group of students had a different instructor, classroom setting, and at least somewhat different learning goals, as well as different means for assessing student performance. A recent study found no correlation between these factors and the pass/fail rate of introductory programming students (Bennedsen & Caspersen, 2007).

In order to extract effectively all of the most relevant information from the initial collected data, both model-based and algorithmic prediction methods were used in the predictive analyses. The predictive contributions of individual, item-level responses were considered, as will as the contributions of combinations of responses and computed values based on those responses. Analytical methods such as regression trees were used, in addition to model-based multiple regression methods, to derive both generalizable and interpretable predictive results, given the available datasets.

Participants completed a paper-based assessment instrument. They were also asked to complete two web-based activities, but the response rate among university and community college students were too low for this data to be usable. These assessments were completed early in the respective courses and responses were compared with later measures of performance in the course (i.e., students' final grades).

The Sample

The target population for this study was introductory computer programming students enrolled in courses taught using a Java-style programming language. Students of many ages might fit that description: high school students might be engaged in computer programming courses in their schools or in external programs; university, four-year college, or community college students might be enrolled in courses as part of their program of study; and adults might be enrolled in introductory programming courses as part of a personal or professional development program. Accordingly, this study included participants from multiple institutions and participants included members of three representative groups. Computer science programs at educational institutions in and near Albany, New York were contacted and solicited for participation in this study.

Participants in a summer introduction to computer science course for precocious (high performing) high school students also were recruited for inclusion in this study[2]. The summer course consisted of approximately 30 students aged 12-16 years who achieved SAT or ACT scores placing them on at or above the mean of college bound high school seniors. The students were enrolled in a three-week summer program for gifted youth offered by Johns Hopkins University. The researcher taught this course. Because of the researcher's role as instructor, the initial assessments were not scored or reviewed until after the course was completed, in order to avoid potential scoring biases.

Two other groups of students were also recruited: those enrolled in introductory computer programming courses using Java-style languages at a university and a community college during the fall 2007 term. These institutions were chosen for their proximity to the researcher's home institution in order to facilitate oversight and administration of the assessment instruments. Students in these courses were typically 18 years of age or older, first year students, although demographics varied depending on the institution's target population.

Data Collection

During the first week of the course, students completed a modified version of Dehnadi's assessment instrument before instruction concerning assignment of values to variables began (see Research Questions, above), so that students' responses were not

[2] See the program description online at http://cty.jhu.edu/summer/catalogs/oscompsci.html

influenced by the course content. The instrument was modified to include several items relating to logic and discrete mathematics, as well as a set of open-ended questions regarding the learners' reasoning. The instrument was designed so that it did not require programming knowledge, but rather the ability to scan and understand examples of assignment of values to variables. The initial instrument was modified to include biographical questions including age, gender, background, etc.; questions focused on logic, permutations, and assignment of values to variables; and open-ended questions that asked the participant to reflect on the task and their own reasoning. U.K.-centric terms appearing on the cover page of the test were translated to their American counterparts: *A-levels* replaced with *AP Exams*, for example. The original twelve questions from Dehnadi's instrument and their responses appeared unaltered. Completed assessments were returned to the researcher for scoring. Results were not made available to the course instructors, in order to prevent any unintentional bias in grading of course assignments.

Students were assigned the task of completing two web-based activities as a homework assignment, in order to minimize the amount of in-class time used by this study. These activities included *Mastermind®* and Sudoku (described in Appendix F. *Mastermind®* Game and Appendix G. Sudoku Game, respectively). For the few university and community college participants who completed the online tasks, data about the participant's actions in each game were captured electronically for later analyses including participant's name, start and stop time, and any errors made. Participants were also asked to respond to five open-ended questions at the end of each task as described in Appendix H. Open-Ended Questions. It was hoped that these questions would offer insight into participants' reasoning that might not otherwise be reflected in the quantitative analysis of other response items.

For each online task, the participant was introduced to the game through an abbreviated instructional version of the game (for example, a 4x4 Sudoku board, rather than the full 9x9 board). The full activity was then presented and the participant was asked to complete two complete instances of each game. Start and stop times were recorded electronically for each instance, as well as any moves made during play. Two instances were presented in order to address those who were initially unfamiliar with the rules of each game.

Students' final grades were collected at the end of term and analyzed in conjunction with all data collected to identify which variables or combination of variables served as the best predictor of those final grades.

Data Analyses

Microsoft Excel[3] was used to enter the data and generate the computed values as described above. The R interactive statistical environment[4] was used to analyze the dataset after using the gdata library to read the Excel spreadsheet into R (Warnes & Gorjanc, 2008).

An exploratory data analysis was conducted in order to use effectively the collected data as predictors for student performance. Item-level responses, computed

---

[3] Microsoft Excel version X SR1 on Mac OS X version 10.5.2

[4] R version 2.6.2 on Mac OS X version 10.5.2

values based on those responses, and combinations of both were used as predictor variables for the criterion variable, students' final grades. Item level response variables as well as predictor variables that were derived from this dataset are described in Appendix A. Early-course Variables of Interest. There is evidence in the literature that suggests that prior programming experience and diversity of languages used, as well as consistency of mental models are positively correlated with student performance.

This analysis was designed to reveal which variables served as the most effective predictors of students' final grades either in raw form. For example, variables were combined, transformed using mathematical functions, or omitted from the analyses following initial consideration. Algorithmic methods, such as recursive partitioning, can be especially effective in learning how variables work in combination with one another. Also, the three datasets that correspond to the three instructional contexts were considered to determine whether the differing instructional contexts played notable roles. Exploratory data analyses were conducted in order to reveal details about relationships among the numerous variables.

Algorithmic techniques such as regression trees were examined in an attempt to ensure the most effective use of all predictor information for prediction of students' final grades. These modern techniques generally permit the analyst to go beyond model-based predictions, particularly in the context of studying moderating effects and interactions among the predictor variables with respect to a criterion variable (Breiman, 2001). These techniques draw heavily from the machine learning community, where they have been used to evolve learning solutions for pattern recognition, genetics, and other highly complex decision algorithms (Breiman, 2001).

Human Subjects Review

Institutional Review Board (IRB) approval was obtained from the University at Albany. Permission for the participation of students at each participating institution was sought and granted. Measures of student performance and demographic data were collected as part of this study and stored on the researcher's computer and backup media. Collected data was secured and maintained so as to protect the exposure of students' data to unauthorized individuals. Since students may receive failing grades in the courses, inadvertent exposure of collected data posed a minimal risk of harm to the student participants in the form of social embarrassment. Participating students were asked to sign and return one copy of the informed consent/assent form and parents of minors were asked to sign and return a parental permission form (see Appendix I. Informed Consent Form, Appendix J. CTY Informed Assent Form, and Appendix K. CTY Parental Permission Form).

The Office of Institutional Research at the University at Albany was notified of research being conducted within the University and appropriate consent obtained from the Computer Science Department Chair and the course instructor. No data was collected about the instructor or his or her instructional methods. No observations were made of the class in-progress. As a result, there was only minimal risk of harm to the department and to the instructor in the event of exposure of collected data.

Chapter 4. Data Analysis

As described above, the purpose of this research was to collect information about novice computer programming students enrolled in introductory computer programming courses as early in each course as possible and to use that information effectively to predict their final numeric course grade.

In this section I highlight pertinent aspects of the sample statistics and then use linear models and recursive partitioning to generate regression trees in various attempts to describe the information content in these data with interactions as defined within each of the regression trees presented. All analyses were performed using the R statistics package (R Development Core Team, 2008). A typical interactive session with R is included in Appendix M.

Characteristics of the Sample

Approximately 175 students were invited to participate in the study of which 144 signed the assent/consent to participate form and submitted the in-class, paper-based assessment instrument. The assessment instrument was administered during the first week of classes in each instance. Eight students withdrew early in the semester necessitating their elimination from the sample, resulting in 136 subjects in the sample consisting of 24 (17.6%) females and 112 (82.4%) males. The sample consisted of 30 (22.1%) high school-aged students taking a summer computer science enrichment course, 35 (25.7%) community college students enrolled in an introductory programming course, and 71 (52.2%) students from a 4-year university enrolled in an introduction to programming course for computer science majors. A cross-tabulation of respondents by institution and gender is provided in Table 7. (Note: Table 7 through

Table 17 summarize the descriptive information referenced in this section and can be found in Appendix L.)

Of the 136 respondents, 80 (58.8%) reported at least some prior programming experience in languages including calculator scripting languages, visual basic (VB), C, C++, Java, JavaScript, PHP, and Pascal. A cross-tabulation of prior programming experience by programming language and gender is provided in Table 8.

A summary of descriptive statistics for continuous variables is provided in

Table 9. Descriptions and coding information for variables is provided in Appendix A. Early-course Variables of Interest.

The criterion variable for this study is students' final numeric course grade in the computer programming course taken, represented as *finalnumeric* in this dataset. Final course grades ranged from 18.26 to 100 on a scale of 100 with a sample mean of 75.12 and a standard deviation of 20.23.

The paper-based assessment instrument also included several items for which responses were categorical, rather than continuous. These are presented in Table 10 through Table 15.

Questions 7 through 18 of the paper-based assessment instrument paralleled questions from Dehnadi's assessment instrument as they concern assignment of values to variables. Each question presented respondents with 10-18 possible choices with the option for respondents to provide a novel response not included in the prepared responses. For coding purposes, responses were numbered from 1 to 18. Respondents who chose to provide novel answers were coded as "0" for record keeping purposes. Additionally, some respondents indicated multiple responses to one or more questions. These multiple responses were coded as "98", if and only if all of the responses for that question indicated that variables were of equal value (e.g., a=5, b=5, c=5). Otherwise, such multiple responses were coded as "99". A tabulation of the response given for each question is provided in Table 16.

Using Dehnadi's scoring guide, item level responses were interpreted as evidence of respondents' use of mental models of the assignment of values to variables. For example, for question 7, a response of 4 indicated the use of mental model 2. If a given mental model was used by a respondent 8 or more times, that respondent is said to have a consistent mental model at the C0 level. The number of respondents who demonstrated mental model consistency at the C0 level is shown in

Table 17, reported by the mental model used and gender. The C0 variable codes for which, if any, mental model was used consistently by the respondent. For example, if the count of M2 instances for a respondent was greater than or equal to 8, C0 would equal 2. Thus, a C0 coding of 0 indicates no consistent mental model was demonstrated.

An additional computed variable was introduced, *c0b*. This variable is a binary variable indicating whether a student had demonstrated use of a consistent mental model, consistent with Dehnadi's use of C0. This variable was coded 0 iff no consistent mental model use was demonstrated and 1 otherwise; preliminary analyses had shown there was no predictive value in this key variable beyond this distinction..

Details of the meaning of each mental model are provided in Figure 3. For the purposes of this study, mental model 2 is the "correct" mental model for assignment of values to variables in the programming languages used in participating courses.

Recursive Partitioning and Regression Trees

Recursive partitioning is an analytical method that algorithmically explores a set of variables in order to find the optimal use of information in those variables (Breiman, 1998). For a regression tree, scores on a criterion variable (here, *finalnumeric*) are sorted from small to large. Recursive partitioning is used at the outset to learn which independent variable best splits the criterion scores so that 'low' scores are distinguished from 'high' scores; all predictors, both categorical and continuous, are examined by the tree algorithm. The split point for the criterion variable is also determined by the algorithm. This process is then repeated for a series of binary splits, such that within each criterion subgroup (say 'low') another split is sought, based on all independent variables (possibly including the one used at the outset) so that more distinctions can be obtained among criterion subgroups; this is continued recursively until no further partitioning 'worthwhile' for predicting the criterion. Various different methods can be employed within the algorithm to define what is meant by the term 'worthwhile.' The resulting structure may be visualized as a binary regression tree, that is a hierarchy, with splits representing each bifurcation.

Tree results are invariant with respect to monotonic transformations of the continuous independent variables; for example the tree would not change if each continuous variable were exchanged for its square root or logarithmic counterpart. Furthermore, trees automatically provide information on how independent variables interact with one another (or with themselves) for prediction of the response variable. At any particular point in the creation of a branch in a tree, all variables, including those that have been used above that branch, are considered by the algorithm for splitting the response set. When predicted values are computed from a completed tree with *k* leaves, the predicted values of the response variable, based on the predictors, consist of the means of the response scores in respective *k* subgroups that correspond to those leaves. The example shown in Figure 7 introduces this idea most simply through the use of a single independent variable with one binary split. Notice that to specify "$c0 < 1$" on the top line is to say that scores on the *c0* variable < 1 were obtained by 71 students with a mean score of 68.46. Further, there were 65 students with *c0* scores > 1 with a mean of 82.40. The latter two means are the predicted scores for this tree (repeated 71 times and 65 times, respectively); these two subgroups have been 'optimally' separated by the regression tree.

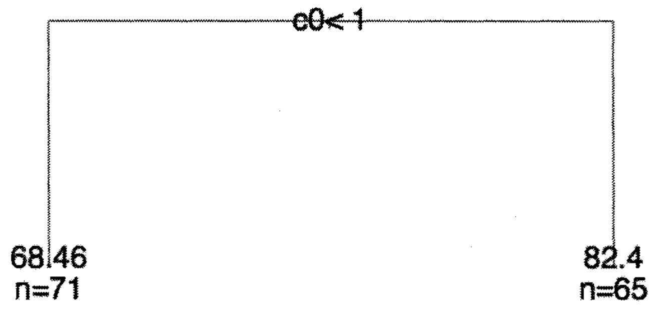


Figure 7: Regression Tree, where (finalnumeric ~ c0)

The one variable regression tree in Figure 7 parallels Dehnadi's research question: does consistency of mental model differentiate students with low final numeric grades from students with high final numeric grades. As we can see from the regression tree, absence of a consistent mental model (indicated as $c0 < 1$) identifies 71 students who received low final numeric grades (mean 68.46) and 65 students who received high final numeric grades (mean 82.4) in their introductory programming courses. The correlation coefficient for this model is 0.34 ($R^2 = 0.12$) suggesting that accounting for additional information about initial student performance may be beneficial.

Figure 8 presents a maximal model that incorporates, in addition to *c0,* other variables such as gender and institution[5], the length of responses to qualitative questions on the paper-based assessment instrument, the length of time spent completing the instrument, the number of valid responses to the Dehnadi-style questions, and a computed mental model variable, *m2.m10.ne*.

[5] Institution is a categorical variable coded by R as: a=CTY, b=community college, c=university.

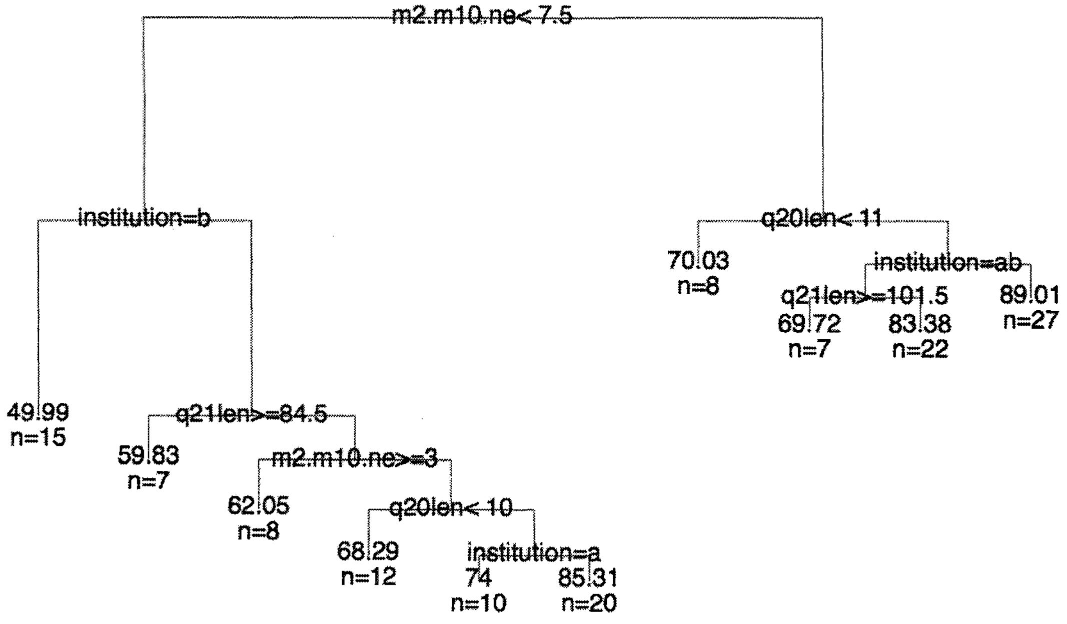


Figure 8: Regression Tree, where (finalnumeric ~ c0 + gender + institution + m2.m10.ne + valid + q19len + q20len + q21len + q22len + durationmins)

The variable *m2.m10.ne* was computed as the sum of instances of evidence for use of mental model 2 (the "correct" Java-style mental model), mental model 10 (interpretation of statements as simple equations with a single answer given), and coded category 98 or "ne" (which is the same as mental model 10, but with multiple responses given). Early exploratory analysis suggested that students with a consistent mental model, either model 2 or model 10, tended to have higher final numeric grades than those with consistent mental models in other model categories as shown in Table 2. All other mental model types were negatively correlated with *finalnumeric*. Note that each of the latter variables correlated positively with the *finalnumeric* criterion. Thus, the computed variable— *m2.m10.ne*—was used to replace these individual variables and incorporated in the independent variable set so that information in these additionally names variables would become part of the mental model variable used for predicting final numeric grade.

Table 2. Correlation matrix for positively correlated mental models

	finalnumeric	**m2**	**m10**	**ne**	**m2.m10.ne**
finalnumeric	1.00	0.22	0.18	0.08	0.28
m2	0.22	1.00	0.31	-0.26	0.84
m10	0.18	0.31	1.00	-0.14	0.47
ne	0.08	-0.26	-0.14	1.00	0.25
m2.m10.ne	0.29	0.84	0.47	0.25	1.00

Inclusion of the above variables presents a more intricate regression tree, making interpretation more difficult. However, the correlation coefficient for this model is 0.64 ($R^2 = 0.40$), representing an improvement over the simpler one variable model of Figure 7.

Several terms in the partition formula were unused by the recursive partitioning algorithm, namely *c0*, *gender*, *valid*, *q19len*, *q22len*, and *durationmins*. These variables were ignored by the tree algorithm because, in relation to the variables seen in the tree, these variables did not improve the predictions. Of special note is the fact that variable *c0* has been ignored in preference to *m2.m10.ne*.

This means that the tree of Figure 8 could have been generated using the simplified formula defined by

finalnumeric ~ institution + m2.m10.ne + q20len + q21len

The *m2+m10+ne* variable (represented in the tree as *m2.m10.ne*) enters the tree at the root with a critical value of 7.5. Subsequent splits rely upon the institutional affiliation of the respondent and the length of responses to qualitative questions. Note that the tree uses the categorical variable *institution* three times to split the subgroups. The *institution=b* branch shows that community college students (split to the left) tend to have the lowest final numeric grades. Within the tree, on the right hand side, the *institution=ab* branch shows that CTY and community college students (split to the left) tend to have lower final numeric grades than university students (shown in the split to the right).

Figure 9 shows a revised tree with a new computed variable, *cc*, introduced to allow for comparison of the community college group to the CTY and university groups, since most of the variability in *institution* was related to community college students.

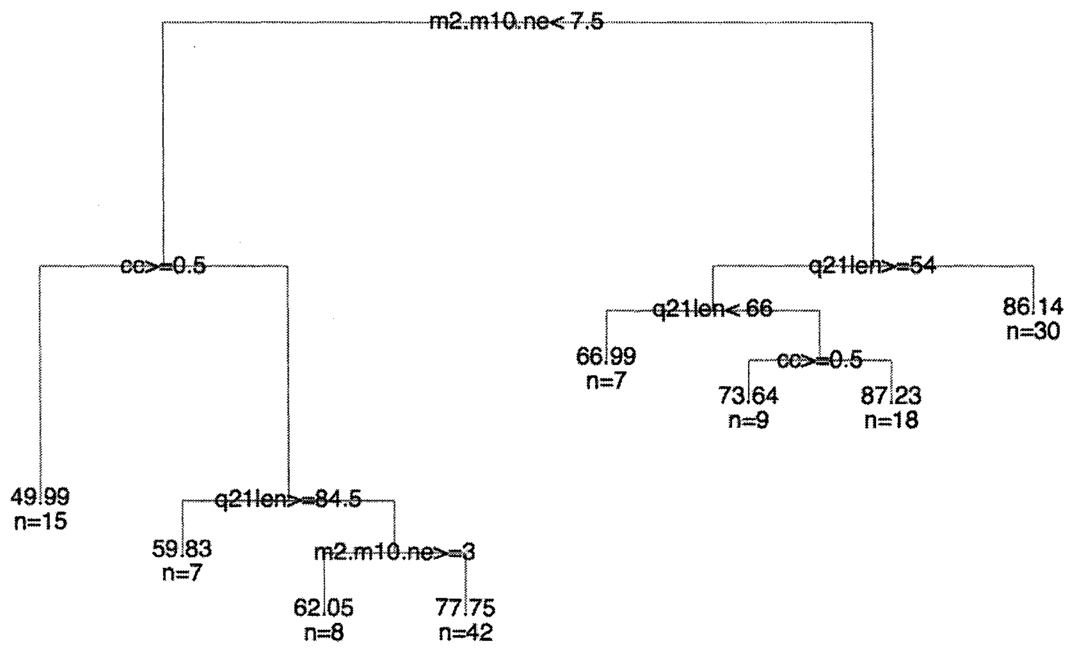


Figure 9: Principal Regression Tree, where (finalnumeric ~ cc + m2.m10.ne + q21len)

Of interest is to what extent institution separated final numeric scores, compared to the other numeric variables. The community college group (indicated by "*cc* < 0.5" in the principal regression tree, above) tended toward both the lower end (mean 49.99) and middle (mean 73.64) of the final numeric grade range. While representing a slight loss of predictive accuracy over the regression tree of Figure 8, correlation is still quite high with a correlation coefficient of 0.60 ($R^2 = 0.36$).

Consistency of mental model (some combination of model 2, model 10, and ne) remains paramount in this regression tree. For students without consistent mental models (as evidenced by their scores on the variable *m2.m10.ne*), those at the community college tended to score lower than those at other institutions. For students with consistent mental models (*m2.m10.ne*), those who provided more concise responses to qualitative item 21 ("what was the most difficult question?") scored better than those with longer responses.

Comparison of tree-based predictions of final numeric grades with actual final grades

Figure 10 shows a scatter plot of predicted final numeric grades, based on the principal regression tree, versus actual final numeric grades. Study of this plot shows that, despite more or less optimal use of the information in all of the predictor variables, the association is far from a simple linear, straight-line form. Indeed, the correlation between these two variables is 0.60 ($R^2 = 0.36$), indicating that 36% of the variation in *finalnumeric* is predicted by this tree, while 64% of that variance is not predicted from the tree. From one point of view there are two groups of students whose performance were particularly notable in reducing the predictability of the criterion scores. Namely, (a) those who were predicted to have low final numeric grades, but who had high final numeric grades, and (b) those who were predicted to have high final numeric grades, but had low final grades. Detailed examination of these subgroups, not given here, shows few similarities among either the members of group (a) or group (b) in this dataset.

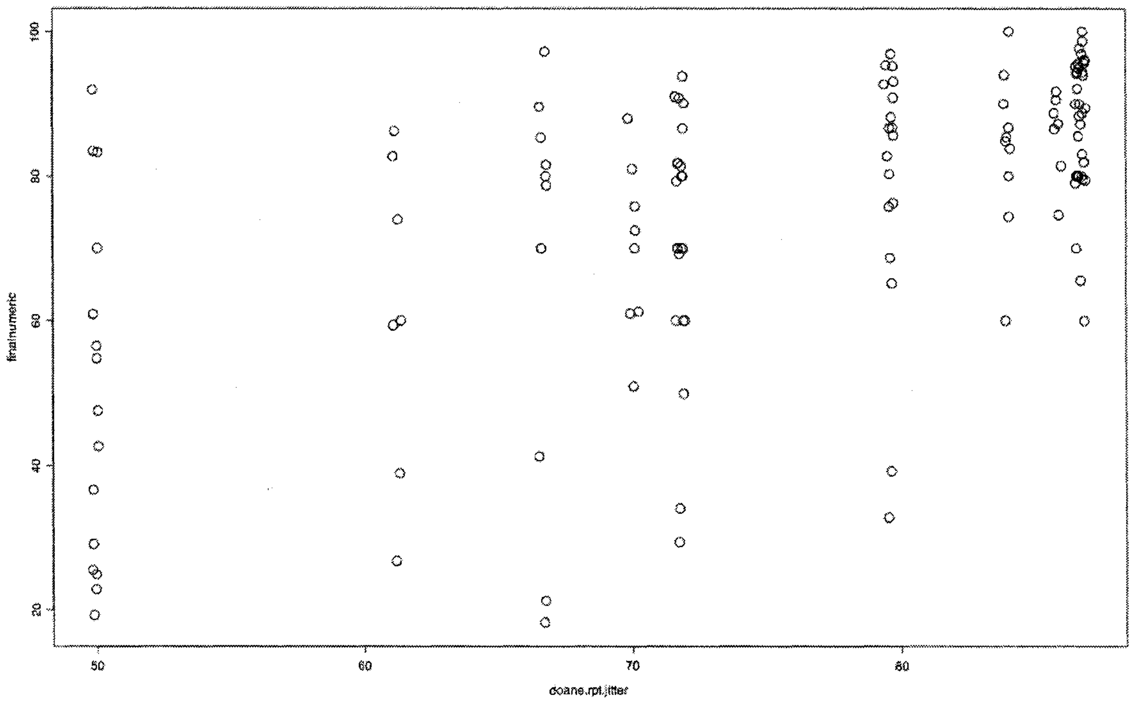


Figure 10: Scatter plot of predicted final grades (based on principal regression tree) versus actual final numeric grades

Comparison of linear models with the Principal Regression Tree

When a linear model that incorporates the same variables as the regression tree of Figure 8 is specified (*finalnumeric ~ c0 + gender + institution + m2.m10.ne + valid + q19len + q20len + q21len + q22len + durationmins*), predictive accuracy is greatly reduced. This main effects linear model yields a multiple correlation coefficient of 0.54 ($R^2 = 0.29$), which is markedly lower than that for the principal regression tree. This is not necessarily surprising since by definition the tree is built upon interaction terms and the main effects (linear) model uses no interactions. But quantification of the difference in predictive usefulness is still helpful. In the linear model, only the terms *m2.m10.ne* and *q21len* are significantly different from zero at the 0.01 level, as indicated in Table 3.

Table 3. Coefficients and t-statistics for main effects linear model; call: finalnumeric ~ c0 + gender + institution + m2.m10.ne + valid + q19len + q20len + q21len + q22len + durationmins

Term	**Estimate**	**Std. Error**	**t value**	**Pr(>|t|)**	
(Intercept)	63.93	7.76	8.24	≈ 0.00	***
c0	0.76	0.63	1.22	0.23	
gender	5.97	4.41	1.35	0.18	
institutionCC	-10.11	5.11	-1.98	0.05	.
institutionUniversity	2.75	4.19	0.66	0.51	
m2.m10.ne	1.26	0.43	2.90	0.00	**
valid	-0.11	0.42	-0.28	0.78	
q19len	0.03	0.04	0.66	0.51	
q20len	0.08	0.04	2.13	0.04	*
q21len	-0.13	0.05	-2.69	0.01	**
q22len	0.06	0.05	1.29	0.20	
durationmins	-0.08	0.27	-0.30	0.77	

Signif. codes: 0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

The main interactions linear model again highlights the significance of the community college affiliation as well as consistency of mental model in the *m2.m10.ne* sense and the length of responses to question 21. It also indicates a significant contribution of *q20len*. In several regression tree analyses, however, the contribution of *q20len* (in terms of interaction) was small when combined with *q21len*; however, this variable added considerable complexity to the interactions so *q20len* was omitted from the following linear model.

Table 4 shows the coefficients and t-statistics for a linear (interaction) model. Terms with no colon (:) are main effects terms. Terms with a single colon (e.g., *cc:m2.m10.ne*) are first order interactions, while the term with two colons (i.e., *cc:m2.m1.ne:q21len*) is a two way interaction. Two way interactions are difficult to interpret in any straightforward way, but the fact that the two way interaction term is at least nominally significant suggests that there is a complicated response surface cutting across the three variables with respect to the criterion, *finalnumeric*.

Table 4. Coefficients and t-statistics for interaction effects linear model; call: finalnumeric ~ cc * m2.m10.ne * q21len

Term	Estimate	Std. Error	t value	Pr(>|t|)	
(Intercept)	80.60	4.19	19.23	≈ 0.00	***
cc	-41.93	8.96	-4.68	0.00	***
m2.m10.ne	0.05	0.56	0.08	0.93	
q21len	-0.14	0.07	-2.00	0.05	*
cc:m2.m10.ne	3.69	1.05	3.51	0.00	***
cc:q21len	0.24	0.13	1.93	0.06	.
m2.m10.ne:q21len	0.01	0.01	1.68	0.10	.
cc:m2.m10.ne:q21len	-0.03	0.01	-1.98	≈ 0.05	.

Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Each of the p-values for the interactions linear model is fairly small, with the exception of the main effects variable *m2.m10.ne*—somewhat unexpected given the role that variable plays in the regression trees. All other main effects and interaction terms are statistically significant. The multiple R-squared value of for the linear (interaction) model is 0.23.

Scatter Plot Matrices for Selected Sets of Variables

Regression trees fail to speak to the question of how members of groups are distributed with respect to some variable of interest. For example, while regression trees offer a predictive model and place each student in one of the prediction categories, they do not distinguish in this context between students with consistent and non-consistent mental models (here, those distinguished by *c0b=0*, versus 1) for variables associated with the principal tree. Scatterplot matrices, especially when enhanced effectively, offers visual information that can add insights about relationships between such variables.

Figure 11 shows an enhanced scatterplot matrix. It has three features in addition to its conventional counterpart: histograms are provided along the diagonal of the matrix; loess regression lines are superimposed on each x,y plot to show predictability of each variable from each other variable, without linear constraints; and points are of two kinds, here distinguished by filled triangles that represent Dehandi's notion of consistency of mental models versus open circles for students who did not demonstrate consistent mental models (using variable *c0b*).

The four variables in this figure include the criterion variable, *finalnumeric*, and three predictors thought to be especially valuable for predicting this criterion. The variable *m2.m10.nej* is a jittered version of *m2.m10.ne*. Jittering adds a small random variable to the base variable in order to offset data points on a plot, thereby making identification of different points easier.

The variable *c0b* is a binary variable indicating whether the student demonstrated a consistent mental model in the Dehnadi sense (it is based wholly on *c0*). Open circles are used to indicate no demonstrated consistency, while filled triangles are used to indicate demonstrated consistency. The points associated with the variable *c0b* distinction show a slight positive correlation between demonstration of a consistent mental model and final numeric grade. However, Figure 11 shows that there is considerable variation across the *finalnumeric* score range of the triangles and circles, so much so that *c0b* is not a particularly good predictor variable.

Figure 11, and especially the loess regression curve, also shows that both *m2.m10.ne* and *q21len* are better predictors at the high end of the final numeric grade range than they are at the low end of the range. Most students either showed no inclination toward using mental models 2, 10, or ne or showed a strong inclination to use those mental models. Relatively fewer students appear to use those models only some of the time.

Generalizability

A variety of attempts were made to examine generalizability properties of the predictive results, separately for trees and linear models. In the end, the decision was made that there are too many idiosyncratic features of the data that were collected to make this enterprise effective. In fact, however, the test statistics associated with the linear model (Table 4) speak to generalizability in one key sense: that is, all predictors, except *m2.m10.ne* (as a main effect) are either statistically significant predictors of the criterion or nearly statistically significant predictors, in the sense that the regression coefficients in the model are statistically different (or near so) from zero.

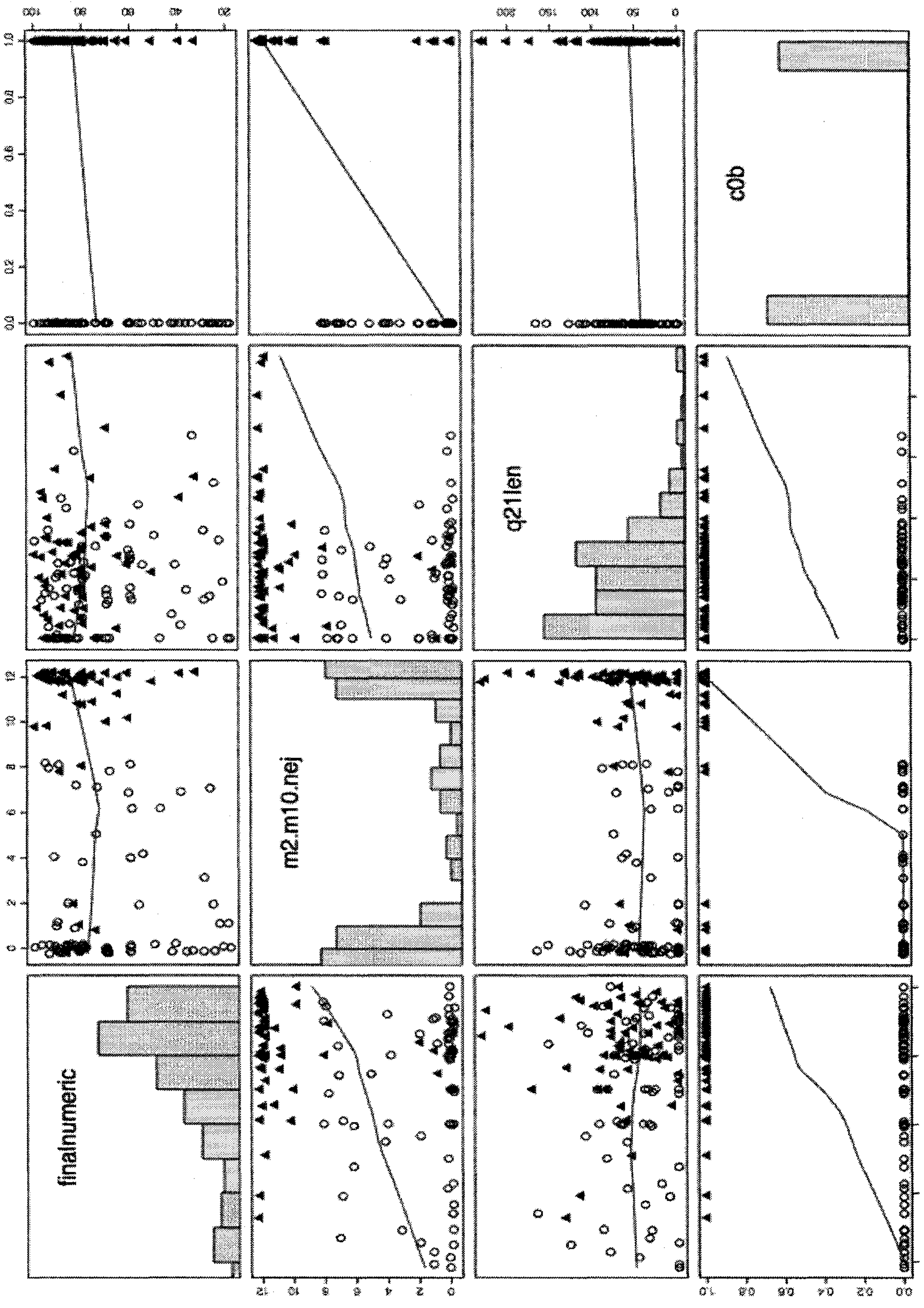


Figure 11: Enhanced scatter plot matrix to highlight effects of Dehnadi's consistency variable

Figure 12 is also an enhanced scatter plot with similar features. Here, however, the two groups highlighted are those at the community college on the one hand (triangles) and those at either CTY or university on the other (circles). Note that the first three variables in Figure 11 and Figure 12 are identical, save for the ways that data points are highlighted. Although the important role of distinguishing between community college students and others has been seen in the summary statistics for both the trees and the linear model analyses. Figure 12 helps make clear that despite notable, even statistically significant differences between community college and other students (above), there are many similarities between these groups of students; this is seen in the way the triangles (for community college students) are broadly distributed in these scatterplots for each of the separate x,y plots when the x's are the 'best' predictors of the criterion.

Figure 16 (found in Appendix N. Enhanced Scatterplot of Selected Variables) shows the relationships among selected variables to enlighten those interested in exploring the dataset further. Figure 17 (found in Appendix O. Enhanced Scatterplot of Linear Model and Principal Regression Tree Predicted Values) shows the correlation between predictions made by the Principal Regression Tree and linear (interaction) model. Again, this is offered to enlighten those interested in the predictive accuracy of the tree-based and linear models.

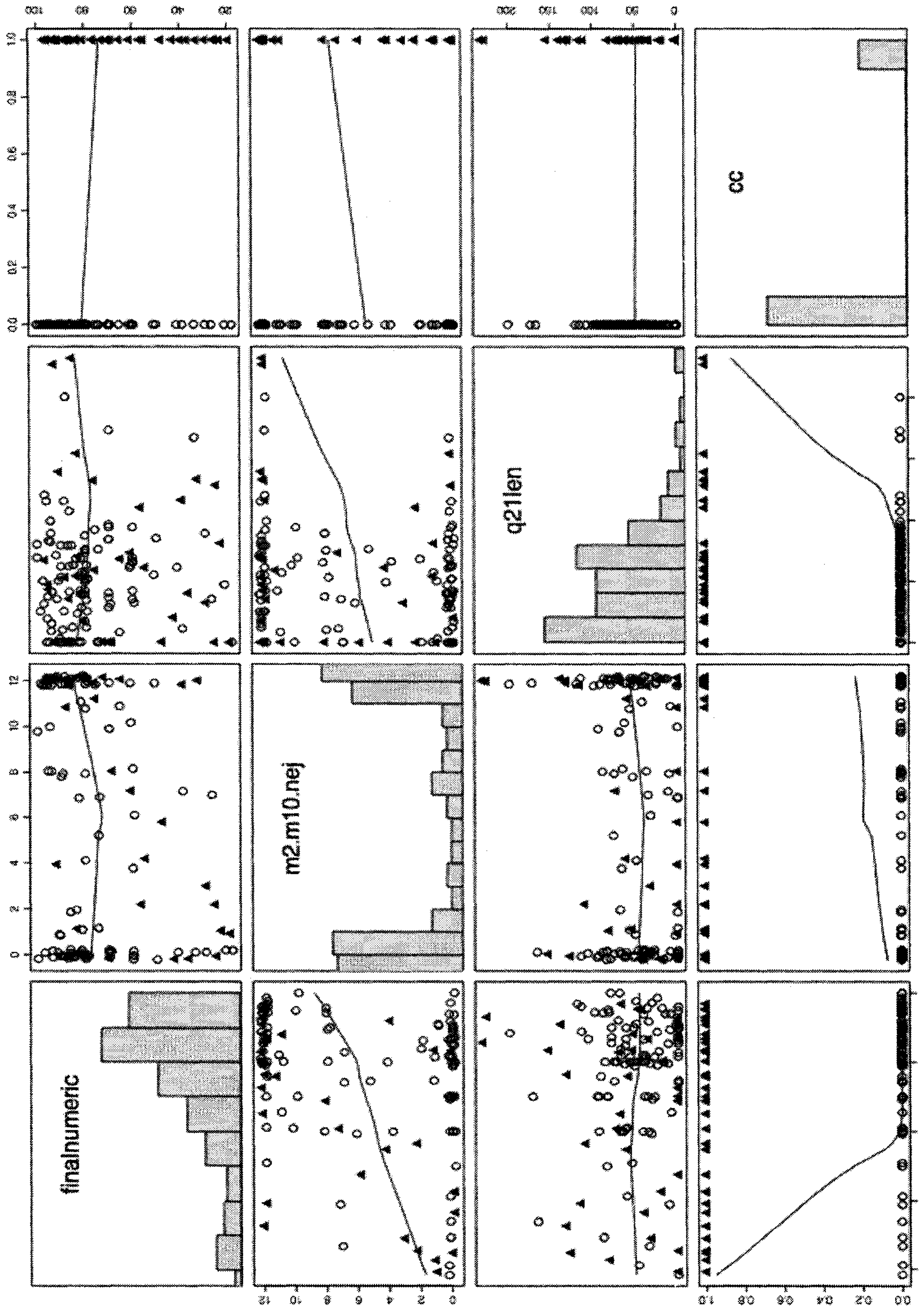


Figure 12: Enhanced scatter plot matrix for variables in the Principal Tree

Chapter 5. Discussion and Conclusions

This study, in part, emulates a study conducted recently in the United Kingdom concerning the mental models of novice programming students (Dehnadi, 2006). The instrument was modified to include several questions relating to logic and discrete mathematics as well as several qualitative questions regarding participants' reactions to the instrument. Item level responses to these questions were considered as possible predictors of student's final numeric grade in introductory computer programming courses.

Additionally, several variables were computed based on responses to the above questions. These computed values were intended to turn qualitative responses into quantitative data, convert multi-valued categorical variables into simple contrast variables, and explore fully the information contained in the item level responses.

Two online, logic-based games were introduced in an effort to identify which, if any, predict final numeric grade in introductory computer programming courses. Unfortunately, response rates to these online games was too low at both the community college and university participating in this study, to make the games data usable. Future studies should strive to ensure that response rates are higher by incorporating game play into class time, offering rewards for task completion, and other structural incentives to encourage participation.

All available data were submitted to both linear model and recursive partition algorithms (Therneau, Atkinson, & Ripley, 2008) in R in order to identify statistically useful variables and simplified models. Based on this analysis, to prediction methodologies were considered and these resulted in the selection of a principal regression tree and linear models with interaction effects that demonstrated reasonable predictability with relatively few independent variables.

The first research question asked: What variables or combination of variables collected at the start of term serve best as predictors of students' final numeric grades at the end of term?

This question is straightforward to answer. Based on the analyses of both the tree-based and linear models the variables *cc*, *m2.m10.ne*, and *q21len* serve as the best predictors of final numeric grade. They are best used in interaction models (either tree-based or linear models).

The second research question stated: Following Dehnadi, is the measure of consistency of students' mental models regarding assignment of values to variables at the outset of the course positively correlated with students' final grades at the end of the course? And, if so, how strong is that relationship?

Both the tree-based and linear models reflected the importance of *m2.m10.ne* as a predictor of *finalnumeric*, supporting Dehnadi's contention that there exists a critical level of consistency below which students tend to have lower numeric grades. Additionally, the critical value of this variable as determined by the recursive partitioning algorithm (7.5) is consistent with Dehnadi's consistency being defined as 8 or more instances where a given mental model was demonstrated (see Figure 8 and Figure 9).

Contrary to what Dehnadi found, however, the simple use of his variable c0 (here, *c0b*, the binary variable used to index consistency or not) did not result in strong or clear prediction of the final numeric grades. In fact, the variable *c0b* was superseded by the variable *m2.m10.ne* (that incorporated more mental models) in both the tree-based and linear models; and interactions with this variable were especially notable.

The most important variable, in terms of statistical prediction, was *m2.m10.ne*, which was a better predictor than *m2* alone. There did not appear to be any notable gender differences, given the available data.

The third research question stated: Are there additional predictor variables that tend to moderate or condition the relationship(s) between the main 'consistency' measures and the outcome measure? If so, what is the nature and extent of the moderator effects?

Gender was recorded and used in preliminary analyses of both kinds, but in all cases had a statistically smaller role to play in predicting the numeric criterion variable than the other, substantive, variables. Total programming courses taken (the variable *totalprgcourses*) was also used in preliminary analyses of both kinds. Again, in all cases, *totalprgcourses* has a statistically smaller role to play in predicting *finalnumeric* than the other variables.

The fourth research question was: What are the similarities and differences among the predictability results across the three main groups: university students, community college students, and (high performing) high school students?

Attendance at community college defined both the low end of the final numeric grade range and mid-range, so that a binary variable that distinguished community college from all other students became especially important in each multiple predictor context. To give an indication of the importance of this variable, a linear (interaction) model that did not include the community college/other distinction yielded a squared multiple R of 0.093, while the counterpart with this variable yielded a squared multiple R of 0.259. The multiple correlations in the two cases were 0.31 and 0.51 respectively.

General discussion

Variables that could be collected during a single class period early in the term were identified and gathered by means of a paper-based assessment instrument incorporating questions about prior programming experience, logic, combinatorics, and assignment of values to variables. This instrument was designed so as not to be a test of prior programming experience. That is, the assessment did not present blocks of programming code for students to interpret. Such an assessment would have amounted to determining whether students already knew the programming language of interest, which while useful to know, is a poor substitute for an assessment that can identify novice programmers who will be likely to struggle with the course. Even so, 17 students explicitly identified confusion over the meaning of INT in Dehnadi's questions as at least a source of uncertainty in their responses. It is unclear that the use of the INT keyword is either necessary or helpful to the questions as presented. Questions 7-18 may well benefit from elimination of the INT keyword, allowing students to focus on the assignment statements themselves, rather than on programming language syntax.

Limitations

Given the wide variety of instructional styles and student learning styles, one might expect that there would be confounding variations of responses and student performance measures with different instructors, programming languages, course formats, etc. However, researchers have reported that the percentage of students failing introductory programming courses remains surprisingly constant, regardless of these factors (Bennedsen & Caspersen, 2007; Dehnadi, 2006; Ma et al., 2007). In short, the question of instructor differences may not be important, so long as grading is rigorous and honest (Dehnadi & Bornat, 2006).

The generalizability of the study was limited by the focus on Java-style programming languages. Other languages, such as LISP and Prolog, use very different models of assignment of values to variables. The mental models and reasoning skills addressed in this study may not have the same import in non-imperative languages. This is taken as an area for future research, rather than a true limitation. What reasoning skills and mental models might play a similar role for those languages remains an open research question.

Also, the various methods of scoring student performance in the three contexts studied (CTY, community college, and university) complicate the interpretation of final numeric grade distributions. A common post assessment activity that addressed the underlying programming concepts addressed in these courses would allow for a more comparable criterion variable.

Finally, the study sample size was constrained based on actual enrollment in courses in the summer and fall 2007 terms. The analysis methods used are known to be robust, even for relatively small sample sizes. The sample size for this study (n = 137) is reasonable for use of recursive partitioning algorithms and regression trees.

Future Research

This study provides the foundation for a wide range of additional research. Several variables were found to be useful in predicting *finalnumeric* when used in interaction models, suggesting that further research on their role and generality is called for.

- **Does first programming language influence the applicability of the study?** To what degree can the same assessment instruments be used to predict students' final grades in programming courses that use non-Java-style languages, such as LISP or Prolog?
- **Does the choice of object-oriented programming language influence the assessment methods?** Do the same assessment methods apply equally well to learners of other object-oriented languages, such as Ruby or C++?
- **What mental models are central for functional programming?** If the assessment instruments do not serve as a predictor when a functional language is the introductory language, then what other mental models and deductive skills, if any could be used as the basis for similar assessment instruments aimed at functional languages, such as LISP?
- **What mental models are central for logical programming?** If the assessment instruments do not serve as a predictor when a logical language is the introductory language, then what other mental models and deductive skills, if any could be used as the basis for similar assessment instruments aimed at logical languages, such as Prolog?

- **Are the assessment instruments affected by cultural differences?** Would the same instruments serve as a predictor in non-English courses?
- **Do the assessment instruments predict students' final grades for high school novice programmers?** As the CS curriculum funnels down into high schools and more high school students take introductory programming courses, will these instruments predict high school students' final grades in those courses?
- **Do the assessment instruments predict students' final grades for community college novice programmers?** Is there an appreciable difference between community college and university novice programmers with respect to their mental models?
- **Do the assessment instruments correlate with AP computer science scores?** Given the current emphasis on Java programming in American AP computer science courses, do the instruments predict students' AP scores?
- **Can we design instructional interventions that will aid students in learning to program?** If the consistency of mental models of assignment are critical to success, can we develop instruction based on improving those mental models, improving their consistency, and so on?
- **Does explicit exposure to mental models improve students' final grades?** Would explicit instruction about the nature of mental models allow students to be more deliberate about their thinking?

As the rate of new computer technology developments increases, new and exciting applications of mental model theory to CSEd also evolve. How might parallel programming, high performance computing, or quantum computation be related to users' mental models of those topics? The range of topics is limited only by the progress of technology and our willingness to explore human understanding.

References

Bayman, P., & Mayer, R. E. (1983). A diagnosis of beginning programmers' misconceptions of BASIC programming statements *Communications of the ACM, 26*(9), 677-679.

Ben-Ari, M. (1998). *Constructivism in computer science education.* Paper presented at the 29th SIGCSE Technical Symposium on Computer Science Education.

Ben-Bassat Levy, R., Ben-Ari, M., & Uronen, P. A. (2003). The Jeliot 2000 program animation system. *Computers & Education, 40*(1), 1-15.

Bennedsen, J., & Caspersen, M. E. (2007). Failure Rates in Introductory Programming. *SIGCSE Bulletin, 39*(2), 32-36.

Blake, M. B. (2006). It's back to school for IT. Retrieved October 26, 2006, from http://news.com.com/Its+back+to+school+for+IT/2010-1011_3-6129515.html

Bonar, J., & Soloway, E. (1989). Pre-Programming Knowledge: A Major Source of Misconceptions in Novice Programmers. In E. Soloway & J. C. Spohrer (Eds.), *Studying the Novice Programmer*. Hillsdale, NJ: Lawrence Erlbaum Associates.

Breiman, L. (1998). *Classification and Regression Trees*: Chapman & Hall/CRC.

Breiman, L. (2001). Statistical Modeling: The Two Cultures. *Statistical Science, 16*(3), 199-231.

Brusilovsky, P., Kouchnirenko, A., Miller, P., & Tomek, I. (1994). *Teaching Programming to Novices: A Review of Approaches and Tools.* Paper presented at the Educational Multimedia and Hypermedia. from http://eric.ed.gov/ERICDocs/data/ericdocs2/content_storage_01/0000000b/80/23/30/ad.pdf.

Chittleborough, G., Treagust, D., Mamiala, T., & Mocerino, M. (2005). Students' perceptions of the role of models in the process of science and in the process of learning. *Research in Science & Technological Education, 23*(2), 195-212.

Clancy, M. (2004). Misconceptions and Attitudes that Interfere with Learning to Program. In S. Fincher & M. Petre (Eds.), *Computer Science Education Research* (pp. 85-100). London, UK: RoutledgeFalmer.

Cunniff, N., Taylor, R. P., & Black, J. B. (1989). Does Programming Language Affect the Type of Conceptual Bugs in Beginner's Programs? A Comparison of FPL and Pascal. In E. Soloway & J. C. Spohrer (Eds.), *Studying the Novice Programmer*. Hillsdale, NJ: Lawrence Erlbaum Associates.

Day, D. L., & Kovacs, D. K. (Eds.). (1996). *Computers, Communication and Mental Models*. London: Taylor & Francis.

Dean, C. (2007, April 17). Computer Science Takes Steps to Bring Women to the Fold. *New York Times*.

Dehnadi, S. (2006). *Testing Programming Aptitude*. Paper presented at the Psychology of Programming Interest Group (PPIG). from http://www.cs.mdx.ac.uk/research/PhDArea/saeed/.

Dehnadi, S. (2007). March 6, 2007 Email. In W. E. J. Doane (Ed.).

Dehnadi, S., & Bornat, R. (2006). *The camel has two humps*. Paper presented at the Little PPIG. from http://www.cs.mdx.ac.uk/research/PhDArea/saeed/.

Denning, P. J., Comer, D. E., Gries, D., Mulder, M. C., Tucker, A., Turner, A. J., et al. (1989). Computing as a Discipline. *Communications of the ACM, 32*(1), 9-23.

Dijkstra, E. W. (1988). On the Cruelty of Really Teaching Computer Science.

Du Boulay, B. (1989). Some Difficulties of Learning to Program. In E. Soloway & J. C. Spohrer (Eds.), *Studying the Novice Programmer*. Hillsdale, NJ: Lawrence Erlbaum Associates.

Du Boulay, B., O'Shea, T., & Monk, J. (1989). The Black Box Inside the Glass Box: Presenting Computing Concepts to Novices. In E. Soloway & J. C. Spohrer (Eds.), *Studying the Novice Programmer*. Hillsdale, NJ: Lawrence Erlbaum Associates.

Duke, R., Salzman, E., Burmeister, J., Poon, J., & Murray, L. (2000). *Teaching programming to beginners - choosing the language is just the first step*. Paper presented at the Proceedings of the Australasian conference on Computing education. from http://portal.acm.org/citation.cfm?id=359381&dl=ACM&coll=portal.

Ehrlich, K. (1996). Applied Mental Models in Human-Computer Interaction. In J. Oakhill & A. Garnham (Eds.), *Mental Models in Cognitive Science: Essays in Honor of Phil Johnson-Laird* (pp. 223-246). East Sussex, UK: Psychology Press.

Fay, A. L., & Mayer, R. E. (1988). Learning LOGO: A Cognitive Analysis. In R. E. Mayer (Ed.), *Teaching and Learning Computer Programming* (pp. 55-74). Hillsdale, NJ: L. Erlbaum Associates.

Fincher, S. (1999). *What are we doing when we teach programming?* Paper presented at the Frontiers in Education Conference.

Fincher, S., & Petre, M. (2004). The Field and the Endeavor. In S. Fincher & M. Petre (Eds.), *Computer Science Education Research* (pp. 1-81). London, UK: RoutledgeFalmer.

Gagné, R. M., & Glaser, R. (1987). Foundations in Learning Research. In R. M. Gagné (Ed.), *Instructional Technology: Foundations* (pp. 49-83). Hillsdale, NJ: Lawrence Erlbaum Associates.

Garnham, A. (1987). *Mental Models as Representations of Discourse and Text.* Chichester, England: Ellis Horwood Ltd.

Gray, W. D., Goldberg, N. C., & Byrnes, S. A. (1993). Novices and programming: Merely a difficult subject (why?) or a means to mastering metacognitive skills? [Review of the book Studying the Novice Programmer]. *Journal of Educational Research on Computers, 9*(1), 131-140.

Guzdial, M. (2004). Programming Environments for Novices. In S. Fincher & M. Petre (Eds.), *Computer Science Education Research* (pp. 127-154). London, UK: RoutledgeFalmer.

Henderson, L. D., & Tallman, J. I. (2005). *Stimulated Recall and Mental Models: Tools for Teaching and Learning Computer Information Literacy (Research Methods in Library and Information Studies).* Lanham, MD: Scarecrow Press.

Johnson-Laird, P. N. (1983). *Mental models: towards a cognitive science of language, inference, and consciousness*. Cambridge, Mass.: Harvard University Press.

Johnson-Laird, P. N. (1988). *The Computer and the Mind: an Introduction to Cognitive Science*. Cambridge, Mass.: Harvard University Press.

Johnson-Laird, P. N. (1989). Mental Models. In M. I. Posner (Ed.), *The Foundations of Cognitive Science* (pp. 469-500). Cambridge, MA: MIT Press.

Johnson-Laird, P. N. (2005). Mental Model and Thought. In K. J. Holyoak & R. G. Morrison (Eds.), *The Cambridge Handbook of Thinking and Reaconsing* (pp. 185-208). New York, NY: Cambridge University Press.

Kahney, H. (1989). What Do Novice Programmers Know About Recursion? In E. Soloway & J. C. Spohrer (Eds.), *Studying the Novice Programmer*. Hillsdale, NJ: Lawrence Erlbaum Associates.

Karsten, R., & Kaparthi, S. (1998). Using dynamic explanations to enhance novice programmer instruction via the WWW. *Computers & Education, 30*(3-4), 195-201.

Kay, R. H. The role of errors in learning computer software. *Computers & Education, In Press, Corrected Proof.*

Kling, R. (1996). *Computerization and controversy: value conflicts and social choices* (2nd ed.). San Diego: Academic Press.

Kuittinen, M., & Sajaniemi, J. (2004). *Teaching roles of variables in elementary programming courses*. Paper presented at the Proceedings of the 9th annual SIGCSE conference on Innovation and technology in computer science education.

Levy, D. (2007). Computer science trouble lies in education, not jobs, Stanford professor says [Electronic Version]. Retrieved April 22, 2007.

Linn, M. C. (1995). Designing computer learning environments for engineering and computer science: The scaffolded knowledge integration framework. *Journal of Science Education and Technology, 4*(2), 103-126.

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. *ACM SIGCSE Bull., 36*(4), 119-150.

Lorenzen, T., & Chang, H.-L. (2006). MasterMind®: A Predictor of Computer Programming Aptitude. *SIGCSE Bull., 38*(2), 69-71.

Ma, L., Ferguson, J., Roper, M., & Wood, M. (2007). *Investigating the viability of mental models held by novice programmers*. Paper presented at the Proceedings of the 38th SIGCSE technical symposium on Computer Science Education. from http://doi.acm.org/10.1145/1227310.1227481.

Margolis, J., & Fisher, A. (2003). *Unlocking the Clubhouse: Women in Computing*. Cambridge, MA: MIT Press.

Mayer, R. E. (1981). The Psychology of How Novices Learn Computer Programming. *ACM Computing Surveys, 13*(1), 121-141.

Mayer, R. E. (1988a). Introduction to Research on Teaching and Learning Computer Programming. In R. E. Mayer (Ed.), *Teaching and Learning Computer Programming* (pp. 1-12). Hillsdale, NJ: L. Erlbaum Associates.

Mayer, R. E. (Ed.). (1988b). *Teaching and Learning Computer Programming*. Hillsdale, NJ: L. Erlbaum Associates.

Mayer, R. E., Dyck, J. L., & Vilberg, W. (1986). Learning to program and learning to think: what's the connection? *Communications of the ACM, 29*(7), 605-610.

McBride, N. (2007). The Death of Computing. Retrieved 2/4/2007, 2007, from http://www.bcs.org/upload/amaxus_pdf/amaxus_conWebDoc_9662.pdf

McKenna, P. (2000). Transparent and opaque boxes: do women and men have different computer programming psychologies and styles? *Computers & Education, 35*(1), 37-49.

Milne, I., & Rowe, G. (2002). Difficulties in Learning and Teaching Programming—Views of Students and Tutors. *Education and Information Technologies, 7*(1), 55-66.

Papert, S. (1980). *Mindstorms: children, computers, and powerful ideas*. New York, NY: Basic Books.

Pea, R. D. (1986). Language independent conceptual 'bugs' in novice programming. *Journal of Educational Computing Research, 2*(1), 25-36.

Perkins, D. N., Hancock, C., Hobbs, R., Martin, F., & Simmons, R. (1989). Conditions of Learning in Novice Programmers. In E. Soloway & J. C. Spohrer (Eds.), *Studying the Novice Programmer*. Hillsdale, NJ: Lawrence Erlbaum Associates.

Perkins, D. N., Schwartz, S., & Simmons, R. (1988). Instructional Strategies for the Problems of Novice Programmers. In R. E. Mayer (Ed.), *Teaching and Learning Computer Programming* (pp. 153-178). Hillsdale, NJ: L. Erlbaum Associates.

R Development Core Team. (2008). R: A Language and Environment for Statistical Computing. Vienna, Austria: R Foundation for Statistical Computing.

Robins, A., Rountree, J., & Rountree, N. (2003). Learning and Teaching Programming: A Review and Discussion. *Computer Science Education, 13*(2), 137-172.

Sajaniemi, J., & Kuittinen, M. (2005). An Experiment on Using Roles of Variables in Teaching Introductory Programming. *Computer Science Education, 15*(1), 59-82.

Shannon, C. E. (2001). A mathematical theory of communication. *ACM SIGMOBILE, 5*(1), 3-55.

Smith, D. C., Cypher, A., & Tesler, L. (2000). Programming by example: novice programming comes of age. *Communications of the ACM, 43*(3), 75-81.

Smith, P. A., & Webb, G. I. (1995). *Reinforcing a Generic Computer Model for Novice Programmers*. Paper presented at the Seventh Australian Society for Computers in Learning in Tertiary Education Conference (ASCILITE'95).

Soloway, E., & Spohrer, J. C. (Eds.). (1989). *Studying the Novice Programmer*. Hillsdale, NJ: Lawrence Erlbaum.

Stephenson, C., Gal-Ezer, J., Haberman, B., & Verno, A. (2005). *The New Educational Imperative: Improving High School Computer Science Education*. New York, NY: Association for Computing Machinery.

Therneau, T. M., Atkinson, B., & Ripley, B. (2008). rpart: Recursive Partitioning (Version 3.1-39 for R).

Tucker, A. (1996a). Crisis in Computer Science Education. *ACM Computing Surveys, 28*(4).

Tucker, A. (1996b). Strategic Directions in Computer Science Education. *ACM Computing Surveys, 28*(4), 836-845.

Tucker, A., Deek, F., Jones, J., McCowan, D., Stephenson, C., & Verno, A. (2003). *A Model Curriculum for K-12 Computer Science: Final report of the ACM K-12 Task Force Curriculum Committee*. New York, NY: Association for Computing Machinery.

Turing, A. (1936). On Computable Numbers, with an Application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society, 42*, 230-265.

United States Department of Labor. (2003). *Tomorrow's Jobs*. Retrieved February 15, 2007. from http://www.bls.gov/oco/oco2003.htm.

United States Department of Labor. (2007). *Computer Programmers*. Retrieved February 15, 2007. from http://www.bls.gov/oco/ocos110.htm.

Vegso, J. (2005). Interest in CS as a Major Drops Among Incoming Freshmen [Electronic Version]. *Computing Research News*, *17*. Retrieved February 15, 2007.

von Neumann, J. (1945). *First Draft of a Report on the EDVAC*: University of Pennsylvania.

Warnes, G. R., & Gorjanc, G. (2008). gdata: Various R programming tools for data manipulation.

Weinberg, G. M. (1988). *The psychology of computer programming*. New York, NY: Van Nostrand Reinhold Co.

Wikipedia.org. (2007a). Mastermind® Game Board.

Wikipedia.org. (2007b). SuDoku Game Board.

Wu, C.-C., Dale, N. B., & Bethel, L. J. (1998). *Conceptual models and cognitive learning styles in teaching recursion*. Paper presented at the Twenty-ninth SIGCSE technical symposium on Computer science education.

Xinogalos, S., Satratzemi, M., & Dagdilelis, V. (2006). An introduction to object-oriented programming with a didactic microworld: objectKarel. *Computers & Education, 47*(2), 148-171.

Yuen, A. (2006). Learning to program through interactive simulation. *Educational Media International, 43*(3), 251-268.

Appendix A. Early-course Variables of Interest

All of the following variables are collected during the first week of the course.

Variable	Source	Description	Coding
institution	PAI[a]	Student's institutional affiliation	CTY CC University
cty	Computed	Indicator of CTY affiliation	0: Not CTY student 1: CTY student
cc	Computed	Indicator of community college affiliation	0: Not community college student 1: Community college student
university	Computed	Indicator of university affiliation	0: Not university student 1: University student
age	PAI	Student's self-reported age	Integer
gender	PAI	Student's self-reported gender	0: female 1: male
ap	PAI	The number of advanced placement-level courses taken in	Integer

		high school	
calculator	PAI	Prior experience programming calculators	0: no 1: yes
vb	PAI	Prior experience programming in Visual Basic	0: no 1: yes
c	PAI	Prior experience programming in C	0: no 1: yes
cpp	PAI	Prior experience programming in C++	0: no 1: yes
java	PAI	Prior experience programming in Java	0: no 1: yes
javascript	PAI	Prior experience programming JavaScript	0: no 1: yes
php	PAI	Prior experience programming in PHP	0: no 1: yes
pascal	PAI	Prior experience programming in Pascal	0: no 1: yes
otherprgcourse	PAI	Previous coursework no reflected in *prior programming experience*	Integer
totalprgcourses	Computed	Sum of *calculator*, *vb, c, cpp, java, javascript, php, pascal,*	Integer

		otherprgcourse	
start	PAI	Student's self-reported start time for PAI	Time
q1	PAI	Combinatorics question	Integer
q2	PAI	Combinatorics question	Integer
q3	PAI	Combinatorics question	Integer
q1c	Computed	Correctness of response to q1	0: incorrect 1: correct
q2c	Computed	Correctness of response to q2	0: incorrect 1: correct
q3c	Computed	Correctness of response to q3	0: incorrect 1: correct
q4	PAI	Growth rate of functions question	0: incorrect 1: correct
q5	PAI	Boolean logic question	a, b, c indicating respondent's choice
q6	PAI	Boolean logic question	a, b, c indicating respondent's choice
q7-q18	PAI	Variable assignment questions from Dehnadi's assessment instrument	Integer: 1-18 indicating response; 0 indicating novel response;

			98 indicating multiple responses all equal; 99 indicating multiple responses no pattern
q19len	Computed	Length of open ended response to question 19: “Please describe your reasoning concerning your responses”	Integer
q20len	Computed	Length of open ended response to question 20: “How does this assessment relate to computer programming”	Integer
q21len	Computed	Length of open ended response to question 21: “What was the most difficult question(s)”	Integer
q22len	Computed	Length of open ended response to question 22: “What other comments do you have regarding this assessment”	Integer
quallen	Computed	Sum of q19len, q20len, q21len,	Integer

		q22len	
m1-m11	Computed	The number of responses to q7-q18 that reflect the given model number, per Dehnadi's scoring guide	Integer
c0	Computed	The model number, if any, for which m1-m11 >= 8, indicating consistency of mental model used, per Dehnadi's scoring guide	Integer: 0-11; 0 indicating no consistent model used.
c0b	Computed	Binary variable indicating demonstration of a consistent mental model	0: No demonstration of consistent mental model 1: demonstration of consistent mental model
ne	Computed	The number of responses to q7-q18 coded as "98", indicating selection of multiple responses to a question where the variables in each of the responses was of equal value (e.g., a=5, b=5, c=5)	Integer: 0-12

nn	Computed	The number of responses to q7-q18 coded as "99", indicating selection of multiple responses to a question where no consistent selection method is evident	Integer: 0-12
nr	Computed	The number of non-responses to q7-q18	Integer: 0-12
valid	Computed	The number of valid responses to q7-q18	Integer: 0-12
m2+m10+ne m2.m10.ne	Computed	Sum of evidence for m2, m10, and ne	Integer
mneg	Computed	Sum of m3, m4, m5, m6, m7, m8	Integer
q23	PAI	Student's self-reported end time for PAI	Time
durationmins	Computed	q23 – start, the number of minutes spent completing the PAI	Integer

[a] Paper Assessment Instrument (PAI), administered in class during the first week of classes, before instruction regarding the assignment of values to variables was given.

Criterion Variable

Variable	Source	Description	Coding
finalnumeric	Course instructor	final numeric grade in course	Real number

Appendix B. Overview of Computer Programming

Early digital computers distinguished between the program and the data upon which the program executed. The program was embodied in the hardware of the computer itself with, perhaps, the ability to alter some parameters of the program's execution (von Neumann, 1945). Data, however, were fed into the computer by means of physical switches, tapes, punch cards, and eventually magnetic storage media.

Representations of computer programs and the data upon which they operate are identical (von Neumann, 1945). Each can be represented by encoding them using a binary (two-valued) number system. The standard symbols used in computer science for such a system are zero (0) and one (1). Complex information can be expressed using sequences of zeros and ones taken together. This design is called stored-procedure computing: the hardware is configured primarily to accept input via some physical mechanism. The input consists of not only the data to be processed, but also instructions constituting the program to be executed. This abstraction allows general purpose computers to be built that can then be programmed for specific tasks based only on the software input into the computer.

For such general purpose computers, the question becomes how the program gets read into the computer and what instructions guide the computer to read the program? This is the bootstrapping problem, suggestive of the notion of *pulling oneself up by ones bootstraps*. Practically, the problem is solved in computers by encoding a small, pre-defined *boot loader* in hardware: using integrated circuits, for example. The boot loader knows just enough about the computer's operation to be able to test the system for critical failures (lack of memory, system malfunctions, and so on) and to load the first program, usually an operating system such as Windows, DOS, OS X, or any of hundreds of others. The operating system then takes control of the system and determines which programs to execute, how to load them into the computer's working memory, and how to execute them once loaded.

The computer is following instructions coded at a very low level when executing the boot loader and the operating system. Low-level languages are those that require coding at or near the level of the actual hardware circuit configurations the computer must create in order to execute code. For example, simple mechanical switches being set to *on* and *off* positions might be considered a very low level of programming. Bits— zeros and ones used in the binary number system— stored in the computer's memory core are a level above switches, but still very low level. The set of bits understood by a given computer architecture is referred to as the *machine language* of that architecture. Each computer architecture has a different machine language. That is, each variant of central processing unit assigns a different meaning to sequences of zeros and ones. For example, on a given architecture the sequence 0001 may mean to perform addition, while on another architecture it may indicate subtraction.

Programming by entering strings of bits is an error prone task for many reasons, including programmer fatigue, the different meanings of identical sequences from one architecture to another, the ease with which a bit can be mistakenly entered as the wrong value, and the difficulty of locating such mistakes. In order to speed the programming process, reduce programming errors, and to make programs more easily readable by humans, stored-procedure computers make use of higher-level programming languages to provide instructions to the computer. High level languages have the benefit that they are easily (to the trained eye) readable by humans, often shorter to code than equivalent bit sequences, and are more easily transferred from one architecture to another. This is accomplished using compilers.

Compilers take source code written in a given programming language and convert it to the machine language understood by a given architecture. This is a gross oversimplification of the process, and there may be many intermediate steps involved, but the approximate description accurately captures the salient point: programs can be converted from human readable versions into functionally equivalent machine readable versions using a compiler. For example, the programming statement 'a = 10' is easily understood by anyone familiar with algebra and might be used to represent the assignment of the decimal value 10 to the logical variable 'a'. Therefore, 'a = 10' is a high level statement that must be converted to machine language in order to be executed on a given architecture. (An overview of computer programming is given in Figure 13, below.)

Java-style languages use a model of assignment of values to variables where the value of the right-hand side (RHS) of an equality statement is computed and stored in the memory location referenced by the variable indicated on the left-hand side (LHS) of the statement (for simple value assignment). The RHS keeps its value. Thus, given:

```
int a = 5;
int b = 25;
a = b;
```

the final values of a and b are both 25, under the correct Java-style model. That is, the value of the RHS of the final statement (namely, the value of b: 25) will be stored in the variable ‘a’ and the variable on the RHS (b) will keep its value.

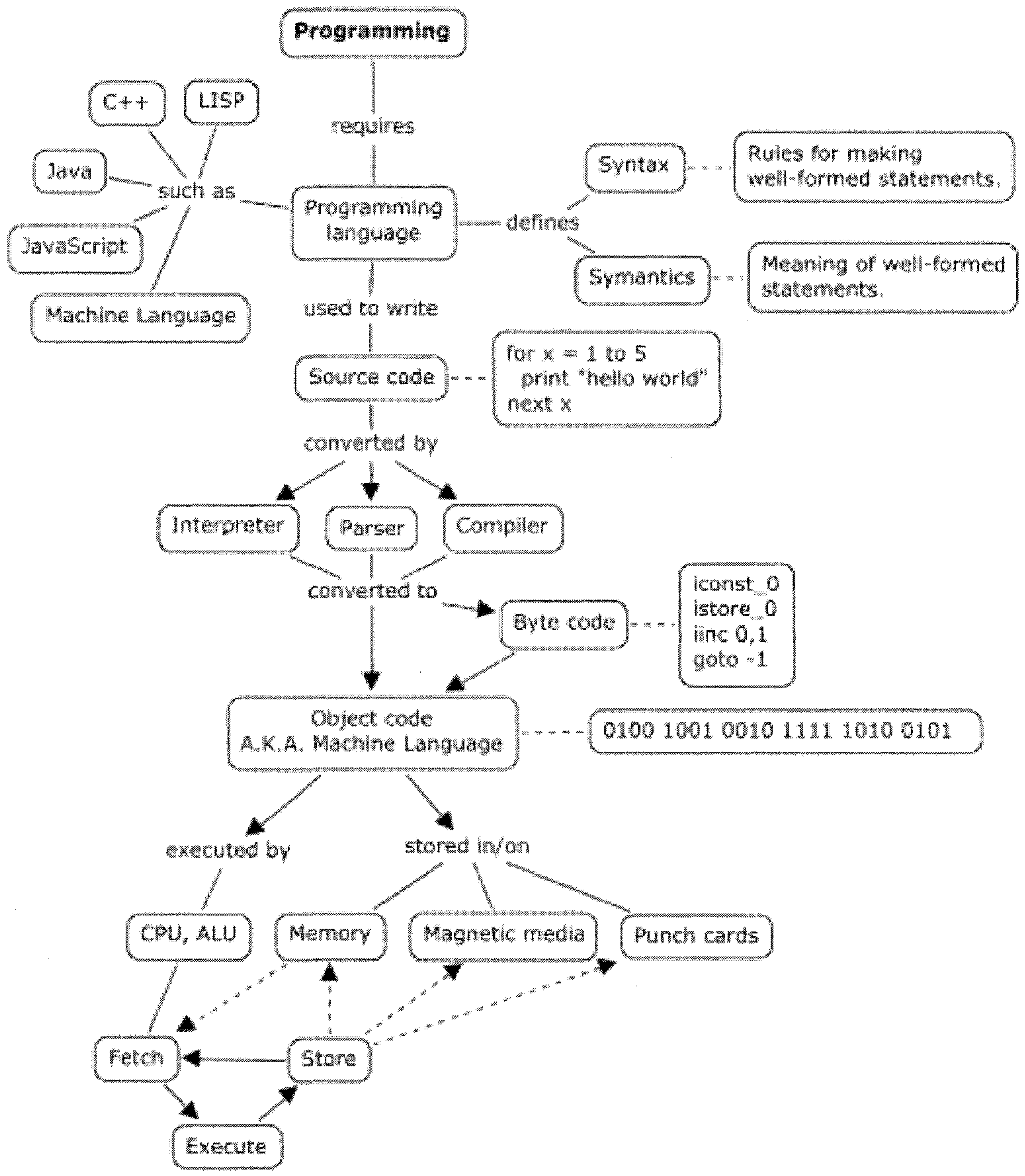


Figure 13: An overview of computer programming

To make the process more concrete, consider a simplified and fictional set of languages presented in Table 5 and Table 6. At the lowest level, they include machine language: bits sequenced so as to encode individual instructions, numeric values, and memory addresses on which those instructions should be carried out. One step higher, mnemonic codes called assembly language are used to encode the same instructions. Using assembly language, the programmer can enter mnemonics that can be converted by the computer automatically into their machine code equivalents by the *assembler*. Finally, high-level instructions are introduced (Table 6). The high-level language can be *compiled* in order to produce the functionally equivalent assembly code or machine language.

Mnemonic (Assembly) Language	**English Description**	**Machine Language**
LDA 5	Load the computer's accumulator with the binary representation of the decimal value 5.	0001 0101
STO 10, 5	Store the binary representation of the decimal value 5 in memory location 10	0010 1010 0101
STA 10	Store the binary representation currently in the computer's accumulator in memory location 10	0011 1010
LMA 10	Load the binary representation	0100 1010

	stored in memory location 10 into the computer's accumulator	
ADD 5	Add the decimal value 5 to the computer's accumulator	0101 0101

Table 5: Fictional assembly-to-machine language translation

High Level Language	**Assembly**	**Machine Language**
a = 5;	LDA 5 STA 10	0001 0101 0011 1010
a = a + 1;	LMA 10 ADD 1 STA 10	0100 1010 0101 0101 0011 1010

Table 6: Fictional high-level-to-assembly-to-machine language translation

Programming using a high level language is generally preferable because the syntax is more human-friendly and easier to understand. Only in cases where direct control of the computer's hardware is necessary is it preferable to program in assembly or machine language.

Appendix C. Modified Dehnadi's Assessment Instrument

Pre-Assessment for Introductory Programming Students

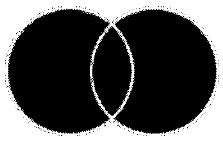
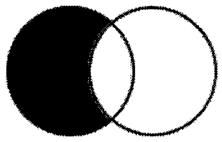
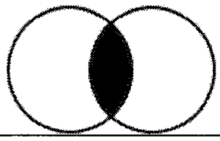
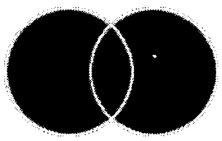
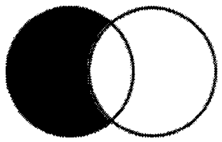
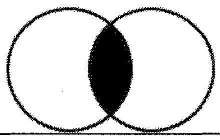
Name	**PLEASE PRINT**
Student Number	
Age	
Gender	M ☐ ☐ F
What **AP-Level** or equivalent courses have you taken, if any?	
What **programming languages** have you written in, if any? (e.g., Java, C++, a macro script, a calculator script, etc.)	
What other **programming courses** have you taken, if any?	
What other **prior experience** do you have that you believe may be relevant to your success in this course?	
Please record the **date and time** you start this assessment:	

Your participation in this study is **optional** and will **not be a factor in the evaluation you receive** in this course.

Please sign below to indicate your willingness to participate in the research project conducted by **William E. J. Doane** during Summer/Fall, 2007.

____
Participant's signature

1. Alice, Bob, and Carol each have a favorite food: donuts, eggs, and fish, in no particular order. List all of the ways in which you can choose two people and two foods. For example: Alice, Bob; donuts, eggs.	**Write your answer in this column**	**Use this column for your rough notes please**
2. You're given 2 dice (die). List all of the possible combinations of rolls that you might get, assuming order doesn't matter. For example: 1,2 and 2,1 are the same combination.		
3. You're given four colored boxes to be stacked on top of each other: red, green, blue, and yellow. Assuming you must stack all four boxes, list each of the ways in which the boxes can be stacked.		

4. The rate of growth of an expression is defined as how quickly the value of the expression increases when increasing values of *n* are used to evaluate the expression. Arrange these 7 mathematical expressions in order from slowest to fastest rates of growth: n^2+2n n^2 n^3 $2n$ n 5 2^n		
5. The dark area of which of these diagrams represents "AND" as in: "MONDAY and PENNSYLVANIA"?	A  B  C 	
6. The dark area of which of these diagrams represents "OR" as in: "MONDAY or NIGHTTIME"?	A  B  C 	

7. Read the following statements and tick the box next to the correct answer in the next column.
 	
`int  a = 10;`
`int  b = 20;`
 	
`a = b;`
	The new values of a and b are:
	
	`[] a = 10      b = 10`
	`[] a = 30      b = 20`
	`[] a = 0       b = 10`
	`[] a = 20      b = 20`
	`[] a = 0       b = 30`
	`[] a = 10      b = 20`
	`[] a = 20      b = 10`
	`[] a = 20      b =  0`
	`[] a = 10      b = 30`
	`[] a = 30      b =  0`
	
	Any other values for a and b:
	`a =      b =`
	`a =      b =`
	`a =      b =`	

8. Read the following statements and tick the box next to the correct answer in the next column.
 	
`int  a = 10;`
`int  b = 20;`
 	
`b = a;`
	The new values of a and b are:
	
	`[] a = 0       b = 30`
	`[] a = 30      b = 10`
	`[] a = 0       b = 10`
	`[] a = 20      b = 0`
	`[] a = 20      b = 20`
	`[] a = 20      b = 10`
	`[] a = 30      b = 0`
	`[] a = 10      b = 20`
	`[] a = 10      b = 10`
	`[] a = 10      b = 30`
	
	Any other values for a and b:
	
	`a =      b =`
	`a =      b =`
	`a =      b =`	

9. Read the following statements and tick the box next to the correct answer in the next column.
 	
`int  big   =  10;`
`int  small = 20;`
 	
`big = small;`
	The new values of big and small are:
	`[] big = 30     small =  0`
	`[] big = 20     small =  0`
	`[] big =  0     small = 30`
	`[] big = 20     small = 10`
	`[] big = 10     small = 10`
	`[] big = 30     small = 20`
	`[] big = 20     small = 20`
	`[] big =  0     small = 10`
	`[] big = 10     small = 20`
	`[] big = 10     small = 30`
	
	Any other values for big and small :
	
	`big =          small  =`
	`big =          small  =`
	`big =          small  =`
		Use this column for your rough notes please

10. Read the following statements and tick the box next to the correct answer in the next column.
 	
`int  a = 10;`
`int  b = 20;`
 	
`a = b;`
`b = a;`
	The new values of a and b are:
	`[] a = 10        b =  0`
	`[] a = 10        b = 10`
	`[] a = 30        b = 50`
	`[] a =  0        b = 20`
	`[] a = 40        b = 30`
	`[] a = 30        b =  0`
	`[] a = 20        b = 20`
	`[] a =  0        b = 30`
	`[] a = 30        b = 30`
	`[] a = 10        b = 20`
	`[] a = 20        b = 10`
	
	Any other values for a and b:
	`a =          b =`
	`a =          b =`
	`a =          b =`	

11. Read the following statements and tick the box next to the correct answer in the next column.
 	
`int  a = 10;`
`int  b = 20;`
 	
`b = a;`
`a = b;`
	The new values of a and b are:
	`[] a = 30       b = 50`
	`[] a = 10       b = 10`
	`[] a = 20       b = 20`
	`[] a = 10       b =  0`
	`[] a =  0       b = 20`
	`[] a = 30       b =  0`
	`[] a = 40       b = 30`
	`[] a =  0       b = 30`
	`[] a = 20       b = 10`
	`[] a = 30       b = 30`
	`[] a = 10       b = 20`
	
	Any other values for a and b:
	`a =         b =`
	`a =         b =`
	`a =         b =`	

12. Read the following statements and tick the box next to the correct answer in the next column.

```
int  a = 10;
int  b = 20;
int  c = 30;

a = b;
b = c;
```

The new values of a and b and c are:

```
[] a = 30   b = 50   c = 30
[] a = 60   b =  0   c =  0
[] a = 10   b = 30   c = 40
[] a =  0   b = 10   c =  0
[] a = 10   b = 10   c = 10
[] a = 60   b = 20   c = 30
[] a = 30   b = 50   c =  0
[] a = 20   b = 30   c =  0
[] a = 10   b = 20   c = 30
[] a = 20   b = 20   c = 20
[] a =  0   b = 10   c = 20
[] a = 20   b = 30   c = 30
[] a = 10   b = 10   c = 20
[] a = 30   b = 30   c = 50
[] a =  0   b = 30   c = 50
[] a = 30   b = 30   c = 30
[] a =  0   b =  0   c = 60
[] a = 20   b = 30   c = 20
```

Any other values for a and b and c :

```
   a =      b =       c =
   a =      b =       c =
   a =      b =       c =
```

Use this column for your rough notes please

13. Read the following statements and tick the box next to the correct answer in the next column.

```
int   a =  5;
int   b =  3;
int   c =  7;

a  =  c;
b  =  a;
c  =  b;
```

The new values of a and b and c are:

```
[] a =  3   b =  5   c =  5
[] a =  3   b =  3   c =  3
[] a = 12   b = 14   c = 22
[] a =  8   b = 15   c = 12
[] a =  7   b =  7   c =  7
[] a =  5   b =  3   c =  7
[] a =  5   b =  5   c =  5
[] a =  7   b =  5   c =  3
[] a =  3   b =  7   c =  5
[] a = 12   b =  8   c = 10
[] a = 10   b =  8   c = 12
[] a =  0   b =  0   c =  7
[] a =  0   b =  0   c = 15
[] a =  3   b = 12   c =  0
[] a =  3   b =  5   c =  7
```

Any other values for a and b and c :

```
   a =      b =       c =
   a =      b =       c =
   a =      b =       c =
```

14. Read the following statements and tick the box next to the correct answer in the next column.
 	
`int a = 5;`
`int b = 3;`
`int c = 7;`
 	
`c = b;`
`b = a;`
`a = c;`
	The new values of a and b and c are:
	
	[] a = 3 b = 5 c = 7
	[] a = 15 b = 10 c = 22
	[] a = 12 b = 8 c = 10
	[] a = 7 b = 7 c = 7
	[] a = 3 b = 5 c = 3
	[] a = 0 b = 0 c = 7
	[] a = 5 b = 3 c = 7
	[] a = 3 b = 3 c = 3
	[] a = 7 b = 5 c = 3
	[] a = 3 b = 5 c = 0
	[] a = 3 b = 7 c = 5
	[] a = 8 b = 10 c = 12
	[] a = 5 b = 5 c = 5
	[] a = 15 b = 8 c = 10
	[] a = 10 b = 5 c = 0
	[] a = 0 b = 0 c = 15
	
	Any other values for a and b and c :
	
	a = b = c =
	a = b = c =
	a = b = c =
		Use this column for your rough notes please

15. Read the following statements and tick the box next to the correct answer in the next column.
 	
`int a = 5;`
`int b = 3;`
`int c = 7;`
 	
`c = b;`
`a = c;`
`b = a;`
	The new values of a and b and c are:
	
	[] a = 15 b = 18 c = 10
	[] a = 7 b = 5 c = 3
	[] a = 7 b = 0 c = 5
	[] a = 0 b = 3 c = 0
	[] a = 10 b = 0 c = 5
	[] a = 5 b = 3 c = 7
	[] a = 3 b = 3 c = 3
	[] a = 12 b = 8 c = 10
	[] a = 7 b = 7 c = 7
	[] a = 15 b = 10 c = 12
	[] a = 7 b = 7 c = 5
	[] a = 8 b = 10 c = 12
	[] a = 0 b = 15 c = 0
	[] a = 7 b = 3 c = 5
	[] a = 5 b = 5 c = 5
	[] a = 3 b = 7 c = 5
	
	Any other values for a and b and c :
	
	a = b = c =
	a = b = c =
	a = b = c =	

16. Read the following statements and tick the box next to the correct answer in the next column.

```
int   a =  5;
int   b =  3;
int   c =  7;

b  =  a;
c  =  b;
a  =  c;
```

The new values of a and b and c are:

```
[] a =  0    b =  7    c =  3
[] a = 12    b =  8    c = 10
[] a = 15    b =  0    c =  0
[] a =  0    b =  7    c =  8
[] a =  3    b =  7    c =  3
[] a =  5    b =  3    c =  7
[] a =  3    b =  3    c =  3
[] a =  7    b =  5    c =  3
[] a = 20    b =  8    c = 15
[] a =  3    b =  7    c =  5
[] a =  5    b =  0    c =  0
[] a =  8    b = 10    c = 15
[] a =  5    b =  5    c =  5
[] a =  8    b = 10    c = 12
[] a =  5    b =  7    c =  3
[] a =  7    b =  7    c =  7
```

Any other values for a and b and c :

```
    a =       b =        c =
    a =       b =        c =
    a =       b =        c =
```

Use this column for your rough notes please

17. Read the following statements and tick the box next to the correct answer in the next column.

```
int  a = 5;
int  b = 3;
int  c = 7;

b =  a;
a =  c;
c =  b;
```

The new values of a and b and c are:

```
[] a =  8     b = 18    c = 15
[] a =  7     b =  0    c =  8
[] a =  5     b =  5    c =  5
[] a = 12     b =  8    c = 15
[] a =  7     b =  0    c =  5
[] a =  3     b =  7    c =  5
[] a =  7     b =  5    c =  3
[] a =  0     b = 15    c =  0
[] a =  0     b =  3    c =  0
[] a =  3     b =  3    c =  3
[] a =  7     b =  7    c =  7
[] a = 12     b =  8    c = 10
[] a =  8     b = 10    c = 12
[] a =  7     b =  5    c =  5
[] a =  5     b =  3    c =  7
[] a =  7     b =  3    c =  5
```

Any other values for a and b and c :

```
    a =       b =        c =
    a =       b =        c =
    a =       b =        c =
```

18. Read the following statements and tick the box next to the correct answer in the next column.

```
int  a = 5;
int  b = 3;
int  c = 7;

a  =  c;
c  =  b;
b  =  a;
```

The new values of a and b and c are:

```
[] a =  0    b = 12    c =  3
[] a =  5    b =  5    c =  5
[] a =  0    b =  7    c =  3
[] a =  8    b = 10    c = 12
[] a = 15    b =  0    c =  0
[] a =  3    b =  7    c =  5
[] a = 12    b = 15    c = 10
[] a =  5    b =  7    c =  3
[] a =  3    b =  3    c =  3
[] a =  7    b =  7    c =  7
[] a = 12    b =  8    c = 10
[] a =  5    b =  0    c =  0
[] a =  5    b =  3    c =  7
[] a =  7    b =  7    c =  3
[] a = 20    b = 15    c = 12
[] a =  7    b =  5    c =  3
```

Any other values for a and b and c :

```
a =     b =       c =
a =     b =       c =
a =     b =       c =
```

Use this column for your rough notes please

19. Please describe your reasoning concerning your responses. How did you arrive at the answers you have given?

20. How does this assessment relate to computer programming?

21. What was the most difficult question(s)? Why?

22. What other comments do you have regarding this assessment?

23. Please record the **time** you finished this assessment:

Appendix D. Dehnadi's Scoring Form

Participant code	Age	Sex	Time to do test	Prior programming	A-Level/s	Prior programming courses	Course result

Questions	Assignment								No effect	Equal sign	Swap values	Remarks (including participants' working notes)
	Assign-to-left		Assign-to-right		Add-Assign-to-left		Add-Assign-to-right		Values don't change (M9) / S	Assign means equal (M10) / S	Swap values (M11) /Ss / I	
	Lose-value (M1) /Ss / I	Keep-value (M2) /Ss / I	Lose-value (M3) /Ss / I	Keep-value (M4) /Ss / I	Keep-value (M5) /Ss / I	Lose-value (M6) /Ss / I	Keep-value (M7) /Ss / I	Lose-value (M8) /Ss / I				
1												
2												
3												
4												
5												
6												
7												
8												
9												
10												
11												
12												
C0												
C1												
C2												
C3												

Additional notes:

s.dehnadi@mdx.ac.uk r.bornat@mdx.ac.uk Simon@newcastle.edu.au

Appendix E. Scoring Guidelines

In the answer sheet for Q1-Q3 (single assignment questions) there are ten single-tick boxes (M1 to M11) and one double-tick box (M10). If the subject gives one tick, we use a single-tick box. If they give two ticks in the positions specified, we use the double-tick box. We can't interpret anything else.

In multiple assignments (Q4 onwards) there is more complexity. First, some of the models are decorated with S or Ss. Instead of just ticking the corresponding model column on the mark sheet, please put the S or Ss next to the tick.

Second, some of the single-tick boxes give alternative models. In this case tick all of the alternative models on the mark sheet. Then, when you've marked all the questions, try to maximise the coherence of the subject's answers by inking in on of the pencil ticks on each row, so as to maximise the numbers in the summary row (labelled C0 on the mark sheet).

Subjective marking is needed to decide what to do with not-entirely- blank scripts. At present we use the following rule:

Rule 1: A consistent response to Q1- Q3 (all the ticks in a single column or in two adjacent columns) can be considered non-blank, but if all we get is three ticks all over the place and nothing else, it's blank. If we could get consistent responses to all the double-assignments or the triple-assignments, then that was non-blank too.

Using joined columns; we can investigate four different levels of consistency in the rows that represent by labels C0, C1, C2 and C3. **Level C0** contents of the 11 single models and demonstrates the highest rate of consistency while sliding toward level C3 leads to lower rate and poorer sign of consistency.

Level C1 contents of 4 columns that each is created by joining two adjacent models, logically carried common concepts. M1 and M2, M3 and M4, M5 and M6, M7 and M8. Each of these new columns logically approved Assignment, assigning value to the left or to the right. **Level C2** contents 2 columns that each is created by joining 4 adjacent models, logically carried common concepts. M1 and M2 and M3 and M4, M5 and M6 and M7 and M8. Each of these new columns logically approved Assignment, assigning value to the left and to the right. **Level C3** contents of a single column that created by joining 8 other models, logically carried common concepts. M1 and M2 and M3 and M4 and M5 and M6 and M7 and M8. The new column logically approved assignment.

Rule 2: Any C level can be considered as subject's level of consistency if:

Mode value in C level >= abs (no. of answered questions * 80%) and

no. of answered questions >= abs (no. of questions * 80%) According to the above rule the subject in the sample is consistent in C1 level. This method creates around 20% flexibility in C level of subject's that answered 80% of the questionnaire.

Appendix F. *Mastermind®* Game

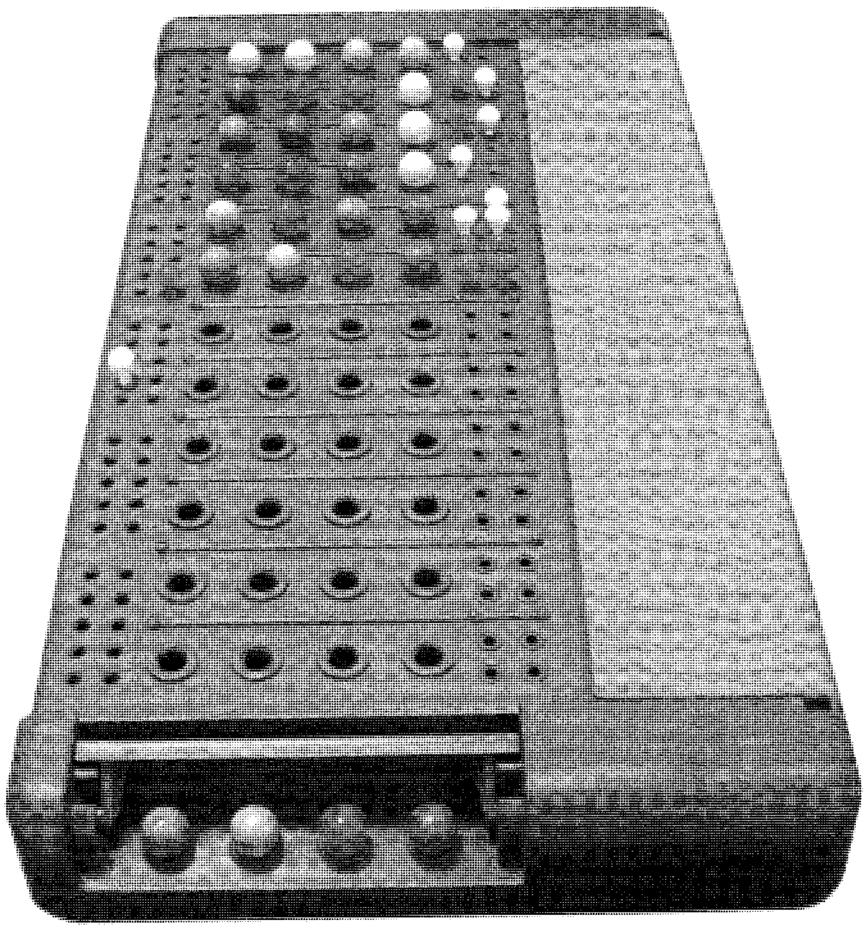
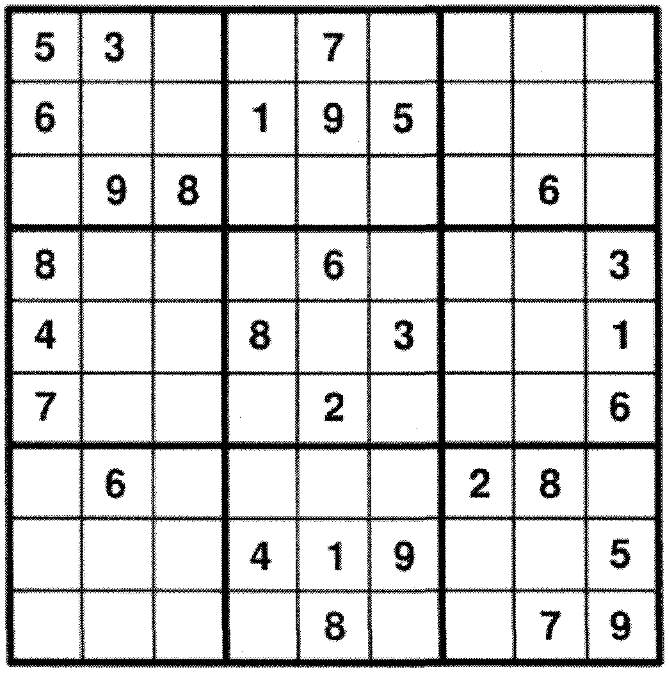


Figure 14: Mastermind Game Board (Source: Wikipedia.org, 2007a)

Mastermind® begins when the codemaker selects a set of colored marbles (with duplication of color allowed) and arranges them in some order (Figure 14, bottom). The codebreaker attempts to guess the colors and order of the marbles. After each guess, the codemaker indicates the number of marbles that match both color and position by placing a red pin to the right of the guess. Guessed marbles that match in color, but not position are indicated using a white pin.

Appendix G. Sudoku Game



5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

Figure 15. Sample SuDoku Game Board (Source: Wikipedia.org, 2007b)

The SuDoku game board presents the player with a set of completed and open squares. Each cell of the board contains a number from 1-9 or is blank, initially. The value to be placed in blank cells is fully constrained by the given values. Players complete the puzzle by deducing which numbers should be placed in which cells. SuDoku has only three rules:

- each row must contain each of the numbers from 1 to 9 exactly once,
- each column must contain each of the numbers from 1 to 9 exactly once, and
- each block (3x3 sub-grid, denoted in Figure 15 by regions bounded by heavy lines) must contain each of the numbers from 1 to 9 exactly once.

These rules impose constraints on the final solution to the puzzle. For example, in the lower-right block, the lower-left cell must contain a 1, since both the row above and the right-hand column already contain a 1.

Appendix H. Open-Ended Questions

The following questions will be included at the end of each web-based activity in order to have participants reflect on their reasoning and the activities.

- Please describe your reasoning concerning your responses. How did you arrive at the answers you have given?
- How does this assessment relate to computer programming?
- What was the most difficult question(s)? Why?
- What other comments do you have regarding this assessment?

Appendix I. Informed Consent Form

Title: Predicting Success In Introductory Computer Programming Courses
Researcher: William E. J. Doane

This study is an attempt to identify students likely to succeed in introductory computer science courses.

You are being asked to participate by completing a **short assessment** at the beginning of your course, completing an **online assignment**, and by allowing your **final grade** in the course to be released to the researcher at the completion of the course. No other effort is require on your part. The tasks are expected to take approximately 60 minutes to complete.

We do not anticipate any risk in your participation other than you may become uncomfortable answering some of the questions.

Although you may not receive direct benefit from your participation, others may ultimately benefit from the knowledge obtained from this research.

Your responses will not be released to your instructor. Your answers and your final grades will be kept **confidential** and will only be able to be traced to you by the researcher. Identifiable grades will not appear in any published work. All information obtained in this study is strictly confidential unless disclosure is required by law. In addition, the Institutional Review Board and University or government officials responsible for monitoring this study may inspect these records.

RESEARCHER	FACULTY SUPERVISOR
William E. J. Doane	Joette Stefl-Mabry
Ph.D. Student	Professor, Information Studies, University at Albany
(518) 810-5427	(518) 442-5120

One copy of this document will be kept together with the research records of this study. Also, you will be given a copy to keep.

If you have any questions concerning your rights as a research participant that have not been answered by the investigator or if you wish to report any concerns about the study, you may contact the University at Albany Office of Research Compliance at (518) 437-4569 or orc@uamail.albany.edu.

Your participation in this project is voluntary. Even after you agree to participate in the research or sign this document, you may decide to leave the study at any time without penalty or loss of benefits to which you may otherwise have been entitled. **If you do not wish to participate, hand in a blank packet.**

I have read, or been informed of, the information about this study. I hereby consent to participate in the study.

____
Please print your name

____
Signature date

Appendix J. CTY Informed Assent Form

Title: Predicting Success In Introductory Computer Programming Courses
Researcher: William E. J. Doane

This study is an attempt to identify students likely to succeed in introductory computer science courses.

You are being asked to participate by completing a **short assessment** at the beginning of your course, completing an **online assignment**, and by allowing your **final marks** in the course to be released to the researcher at the completion of the course. No other effort is require on your part. The tasks are expected to take approximately 60 minutes to complete.

We do not anticipate any risk in your participation other than you may become uncomfortable answering some of the questions.

Although you may not receive direct benefit from your participation, others may ultimately benefit from the knowledge obtained from this research.

Your answers and your final marks will be kept **confidential** and will only be able to be traced to you by the researcher. Identifiable marks will not appear in any published work. All information obtained in this study is strictly confidential unless disclosure is required by law. In addition, the Institutional Review Board and University or government officials responsible for monitoring this study may inspect these records.

RESEARCHER	FACULTY SUPERVISOR
William E. J. Doane	Joette Stefl-Mabry
Ph.D. Student	Professor, Information Studies, University at Albany
(518) 810-5427	(518) 442-5120

One copy of this document will be kept together with the research records of this study. Also, you will be given a copy to keep.

If you have any questions concerning your rights as a research participant that have not been answered by the investigator or if you wish to report any concerns about the study, you may contact the University at Albany Office of Research Compliance at (518) 437-4569 or orc@uamail.albany.edu.

Your participation in this project is voluntary. Even after you agree to participate in the research or sign this document, you may decide to leave the study at any time without penalty or loss of benefits to which you may otherwise have been entitled. **If you do not wish to participate, hand in a blank packet.**

I have read, or been informed of, the information about this study. I hereby agree to participate in the study.

____
Please print your name

____
Signature date

Appendix K. CTY Parental Permission Form

Title: Predicting Success In Introductory Computer Programming Courses
Researcher: William E. J. Doane

This study is an attempt to identify students likely to succeed in introductory computer science courses.

Your child is being asked to participate by completing a **short assessment** at the beginning of his or her course, completing an **online assignment**, and by allowing their **final marks** in the course to be released to the researcher at the completion of the course. No other effort is required on their part. The tasks are expected to take approximately 60 minutes to complete.

We do not anticipate any risk in your child's participation other than that he or she may become uncomfortable answering some of the questions.

Although your child may not receive direct benefit from his or her participation, others may ultimately benefit from the knowledge obtained from this research.

Your child's answers and final marks will be kept confidential and will only be able to be traced to him or her by the researcher. Identifiable marks will not appear in any published work. All information obtained in this study is strictly confidential unless disclosure is required by law. In addition, the Institutional Review Board and University or government officials responsible for monitoring this study may inspect these records.

RESEARCHER	FACULTY SUPERVISOR
William E. J. Doane	Joette Stefl-Mabry
Ph.D. Student	Professor, Information Studies, University at Albany
(518) 810-5427	(518) 442-5120

One copy of this document will be kept together with the research records of this study. Also, you will be given a copy to keep.

If you have any questions concerning your rights or your child's rights as a research participant that have not been answered by the investigator or if you wish to report any concerns about the study, you may contact the University at Albany Office of Research Compliance at (518) 437-4569 or orc@uamail.albany.edu.

Your child's participation in this project is voluntary. Even after you agree to allow him or her to participate in the research or sign this document, you or your child may decide to leave the study at any time without penalty or loss of benefits to which you may otherwise have been entitled.

I have read, or been informed of, the information about this study. I hereby agree to allow my child to participate in the study.

______________________________ ______________________________
Please print your child's name Please print your name

____
Your Signature date

Appendix L. Sample Statistics

Table 7:

Sample Size by Institution and Gender

	CTY	CC	University	*Subtotal by gender*
Female	5	7	12	24
Male	25	28	59	112
Subtotal by institution	30	35	71	**n = 136**

Table 8: Prior Programming Experience by Programming Language and Gender

	Calculator[a]		VB[b]		C		C++		Java	
	no	yes	no	yes	no	yes	no	yes	no	yes
Female	23	1	20	4	24	0	22	2	20	4
Male	95	17	90	22	104	8	80	32	86	26
Subtotal	118	18	110	26	128	8	102	34	106	30

	JavaScript		PHP		Pascal	
	no	yes	no	yes	no	yes
Female	24	0	24	0	24	0
Male	100	12	105	7	111	1
Subtotal	124	12	129	7	135	1

[a] Calculator refers to prior programming experience using calculator scripting languages
[b] VB refers to prior programming experience using Visual Basic

Table 9: Descriptive Statistics

	Min.	Median	Mean	S.D.	Max.	<NA>
finalnumeric	18.26	80.19	75.12	20.23	100.00	
age	12.00	18.00	18.42	4.37	47.00	6.00
ap	0.00	0.00	0.72	1.38	7.00	
otherprgcourse	0.00	0.00	0.13	0.58	4.00	
totalprgcourses	0.00	1.00	1.13	1.47	9.00	
q1	2.00	9.00	12.95	30.87	360.00	3.00
q2	1.00	21.00	24.81	21.76	256.00	5.00
q3	0.00	24.00	20.85	21.91	256.00	5.00
q19len	0.00	57.00	69.10	55.96	336.00	
q20len	0.00	41.50	53.85	56.77	355.00	
q21len	0.00	50.50	54.73	46.35	232.00	
q22len	0.00	0.00	18.15	39.80	314.00	
quallen	0.00	180.00	195.80	152.80	1057.00	
durationmins	9.00	23.00	22.87	6.59	50.00	14.00
ne	0.00	0.00	0.89	2.81	12.00	
nn	0.00	0.00	1.02	2.69	12.00	
nr	0.00	0.00	3.09	4.60	12.00	
valid	0.00	10.50	7.00	5.40	12.00	
m2+m10+ne	0.00	7.00	5.98	5.40	12.00	
mneg	2.00	20.00	18.46	2.85	20.00	

Table 10: Responses to Paper Assessment Instrument Question 1

2	3	4	5	6	8	9	12	16	18	24	27	35	36	360	<NA>
3	4	1	1	8	3	94	1	1	8	3	2	1	2	1	3

Table 11: Responses to Paper Assessment Instrument Question 2

1	3	15	16	17	18	19	20	21	24	30	35	36	38	72	256	<NA>
1	1	5	1	1	3	2	12	81	1	1	2	17	1	1	1	5

Table 12: Responses to Paper Assessment Instrument Question 3

3	4	5	6	7	8	9	11	12	14	15	16	17	18	20	21	22	24	25	26	256	<NA>
1	5	2	7	4	3	4	2	2	1	1	9	1	1	1	1	1	82	1	1	1	5

Table 13: Correctness of Responses to Paper Assessment Instrument Question 4 by Gender

	Incorrect	Correct	<NA>
Female	18	5	1
Male	79	29	4
Subtotal by correctness	97	34	5

Table 14 Responses to Questions 5 by Response and Gender

	a	b	c	<NA>
Female	0	0	24	0
Male	24	5	82	1
Subtotal by response	24	5	106	1

Table 15: Responses to Questions 6 by Response and Gender

	a	b	c	<NA>
Female	16	6	2	0
Male	41	36	32	3
Subtotal by response	57	42	34	3

Table 16: Item Responses for PAI Questions 7 - 18

	0[a]	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	98[b]	99[c]
q7	0	11	1	3	$\boxed{55}$	0	10	6	1	1	0									12	18
q8	0	0	0	0	2	14	10	0	8	$\boxed{54}$	0									12	11
q9	0	1	0	0	6	9	0	$\boxed{56}$	2	11	0									11	15
q10	0	2	6	1	1	1	1	$\boxed{37}$	0	1	8	17								15	16
q11	0	1	$\boxed{39}$	5	3	0	0	1	0	18	0	8								14	16
q12	0	2	0	1	1	4	0	0	1	11	3	2	$\boxed{51}$	0	1	0	3	0	0	11	12
q13	1	1	2	0	0	$\boxed{37}$	8	3	21	2	2	1	0	0	0	0				10	11
q14	1	1	0	1	4	$\boxed{32}$	0	9	2	21	1	1	2	1	0	0	0			8	11
q15	1	0	18	0	0	0	9	$\boxed{36}$	1	5	1	1	0	0	1	0	4			7	8
q16	0	0	1	0	1	2	10	4	18	0	0	0	1	$\boxed{35}$	1	0	3			7	8
q17	1	1	0	2	0	0	1	20	0	0	5	2	1	2	$\boxed{34}$	7	2			7	6
q18	1	1	4	0	1	0	0	0	2	3	2	1	0	7	$\boxed{34}$	0	20			7	7

Note. Boxed entries indicate mental model 2, the "correct" model for Java, C++, and JavaScript.

[a] "0" indicates a single response was given other than those presented to the subject.

[b] "98" indicates multiple responses were given where the variables in each selected item was equal (e.g., a=5, b=5, c=5 and a=10, b=10, c=10).

[c] "99" indicates multiple responses where given where no consistent selection method was evident.

Table 17: C0 consistency of Mental Models Used by Mental Model and Gender

	Mental Model				
	2	**4**	**9**	**10**	*Subtotal by gender*
Female	4	0	1	2	7
Male	46	1	4	7	58
Subtotal by model	50	1	5	9	*Total consistent* = 65

Appendix M. Typical R Session

R is a free, open-source statistics package that allows the user to interact with data using both a command-line and a visual interface (R Development Core Team, 2008). Typically, interactive R sessions consist of setting environment options, establishing a clear workspace, loading support libraries for the types of data analyses and manipulation being performed, loading a dataset, and using the support library to perform the analyses. The R interactive command prompt is indicated by the greater-than symbol (>).

```
> options(digits=3, scipen=1000)
> rm(list=ls()) # remove all objects from current workspace
> library(gdata)
> library(Hmisc)
> library(rpart)
> doane.data = read.xls("doane0402.xls")
> dim(doane.data) # display dimensions of the data
> doane.data[1:4,] # display first 4 rows of data
> attach(doane.data)
> panel.hist <- function(x, ...)
      {
          usr <- par("usr"); on.exit(par(usr))
          par(usr = c(usr[1:2], 0, 1.5) )
          h <- hist(x, plot = FALSE)
          breaks <- h$breaks; nB <- length(breaks)
          y <- h$counts; y <- y/max(y)
          rect(breaks[-nB], 0, breaks[-1], y, col="cyan", ...)
      }
> doane.lm1 = lm(finalnumeric ~ c0 + institution)
> summary(doane.lm1)
> doane.rpt1 = rpart(finalnumeric ~ c0 + institution)
> cor(predict(doane.rpt1), finalnumeric)
> par(xpd=TRUE); plot(doane.rpt1); text(doane.rpt1, use=T,cex=2)
> doane.rpt1.jitter <- jitter(predict(doane.rpt1), 1.2)
> plot(doane.rpt1.jitter, finalnumeric)
> identify(doane.rpt1.jitter, finalnumeric)
> pairs(cbind(finalnumeric, c0, institution, doane.rpt1.jitter),
diag.panel = panel.hist, panel=panel.smooth, pch=1+16*c0b, cex=1.75)
> savehistory("diss.history") # save the R command history
> save.image() # save the R workspace
> q()
```

Appendix N. Enhanced Scatterplot of Selected Variables

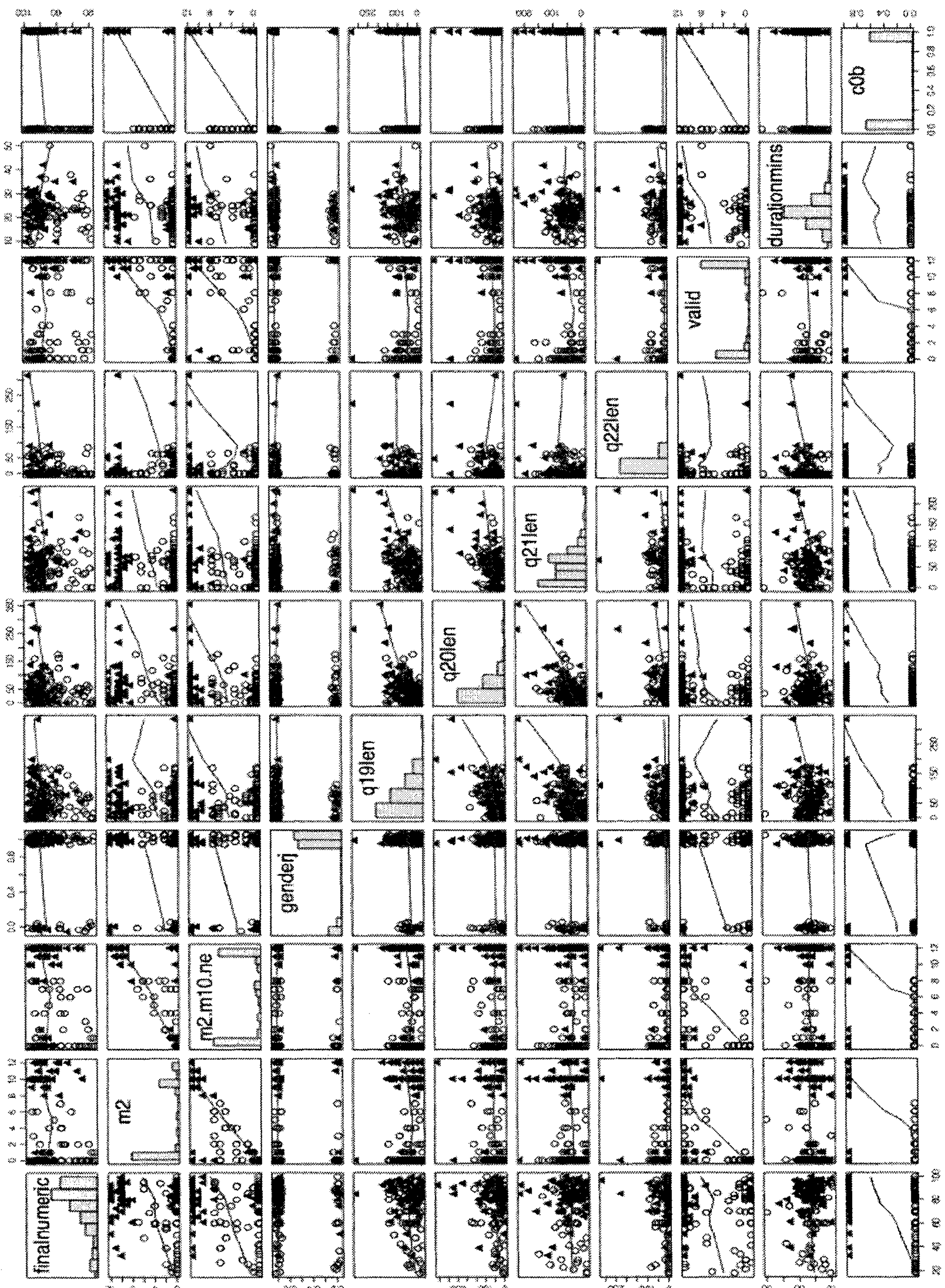


Figure 16: Enhanced Scatterplot of Selected Variables

Appendix O. Enhanced Scatterplot of Linear Model and Principal Regression Tree Predicted Values

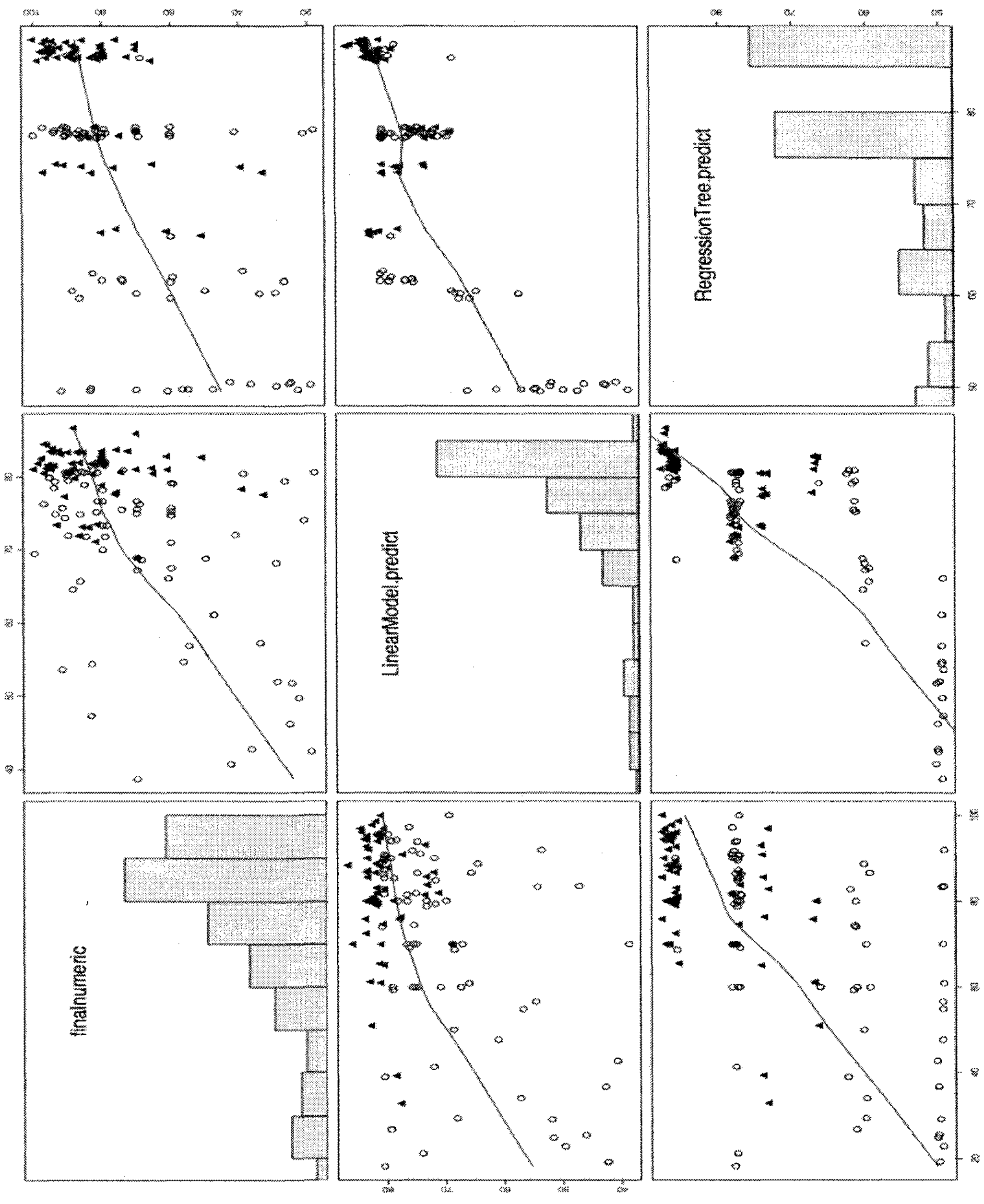


Figure 17: Enhanced Scatterplot of Linear Model and Principal Regression Tree Predicted Values