ON THE ORIGINS OF PROGRAMMERS: IDENTIFYING PREDICTORS OF SUCCESS FOR AN OBJECTS FIRST CS1

by

Philip R. Ventura, Jr.

November 20, 2003

Major Professor: Bina Ramamurthy, Ph.D.

A dissertation submitted to the
Faculty of the Graduate School of
The State University of New York at Buffalo
In partial fulfillment of the requirements for the degree of
Doctor of Philosophy

Department of Computer Science and Engineering

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

UMI Number: 3113537

Copyright 2003 by
Ventura, Philip R., Jr.

All rights reserved.

INFORMATION TO USERS

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleed-through, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

UMI Microform 3113537
Copyright 2004 by ProQuest Information and Learning Company.
All rights reserved. This microform edition is protected against
unauthorized copying under Title 17, United States Code.

ProQuest Information and Learning Company
300 North Zeeb Road
P.O. Box 1346
Ann Arbor, MI 48106-1346

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Copyright by

Philip R. Ventura, Jr.

2003

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Acknowledgments

I wish to thank my advisor, Bina Ramamurthy, most of all for her support of my choice of topic when it was uncertain whether CSEd would be acceptable in our computer science department. Bina has also given me tremendous freedom in exploring possible avenues for my research, as well as, allowed me to complete my work according to my own work-style.

I also wish to thank Svet Braynov, one of my dissertation committee members, for his constant support and encouragement. Svet always read chapters and provided feedback in a timely manner. In general, he was a great supporter of my work, especially the rigorous statistical work. Svet's door was always open, whether it was to discuss my work or to provide advice both in dealing with hardships of writing a dissertation as well as navigating the waters of an academic job search. For this, I will always be grateful.

In addition, I wish to thank Ken Regan, another dissertation committee member, for being a strong advocate from the time I applied to the Department's graduate program. I also thank Ken for his support when I took CSE596 during a time in my life when I was experiencing tremendous personal difficulty. He was incredibly understanding and encouraging. For my dissertation, Ken provided both support and asked questions that

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

led to deeper analyses of the data I had collected. In the end, these analyses contributed much to the dissertation.

I also owe a debt of gratitude to Bill Rapaport, the last of my dissertation committee members, for his keen editorial work. Bill was also a strong advocate within the Department for a CSEd dissertation to be a part of computer science.

I also wish to thank J. Philip East of the University of Northern Iowa for serving as my outside reader.

With regards to the statistical work of my dissertation, there are three individuals who contributed greatly. Michael Zborowski took a great deal of time out of his very busy schedule to offer advice both in the design of the study as well as suggestions for analysis. Greg Wilding of the Biostatistics Consulting Service at UB provided insight into test assumptions. Finally, Ken Levy served as an extra pair of eyes to read the statistical chapters of my dissertation.

I wish to thank my teacher, my boss, my mentor, my colleague and friend, Alistair Campbell. Alistair was the instructor for CSE114 when I took it as an undergraduate prior to enrolling the department's graduate program. I was and still am in awe of his teaching ability. Knowing that I was applying for admittance to the grad program, he was another very vocal advocate of my being accepted. I attribute my admittance, with aid, to the CSE department to support of both Alistair and Ken Regan.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

I owe a debt of gratitude to the UTAs: Dan Britt, Wes Fang, Vince Millioto, Donna Vaccaro, and Bernie Sorenson. They have exceeded every expectation I had for them when I conceived the undergraduate TA program years ago. Their work ethic has been inspirational. Their teaching ability and infinite patience are, in a large part, the reason why students are successful. Further, having the UTAs has meant that as an instructor I no longer needed to worry about the quality of instruction in the recitations.

I want to thank Carl Alphonce for being my "partner in crime" in instituting the objects-first approach at UB. Carl was always open-minded, if not necessarily convinced (initially), about some of the curricular developments. Further, Carl's ability to organize has been a channel for my creative energies.

I'd like to thank Adrienne Decker for her help first as my TA, then as friend and colleague. Adrienne's organizational ability generally ensured that my assignments got graded and that I got to class. Further, the countless lunches including those at the Tiffin Room and a wealth of Indian dinners, as well as, good spirits at TA meetings and exam proctoring have largely helped me to keep my sanity in this endeavor. I have danced at her wedding and now hope to sit at her dissertation defense.

I wish to thank Chris Egert. As the TA in the first graduate course I took at the University, Chris always pushed me to strive for the very best. At the time I hated him for it. However, I realized afterward that it was exactly the kick in the pants I needed. Not only did I come away a firm understanding of operating systems (ok so I still don't get IPC problems), but my coding and debugging skill had also improved exponentially.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

It is said that a Ph.D. is an apprenticeship, and I was Chris's. He provided insight into my work, support when I thought I was defeated, and helped me to build my self-confidence both as an instructor and as a researcher. He is one of my best friends and I am delighted we will continue working together at UWG.

I also need to acknowledge the help and support of various others whom I came to know at UB: Nathan Smith, Sarah Hacker, and John Whitley.

My thanks to the faculty at State University of West Georgia first for hiring me into a Department that I think will be a fantastic fit given my background and aspirations. Also, by offering me a job that required I have a Ph.D., they provided motivation for my finishing.

While there have been many who have supported my work there have of course been detractors, within the Department, the University, and the CSEd academic community. However, addressing their concerns has made my work better. In fact the idea for this dissertation was spawned by critics in the University. For all of them, I say, thank you.

My initial interest in computers came when my parents, Phil and Diane Ventura, bought me my first computer at a time when few had computers in the home. Both my parents support my computer interest through several years. The love, encouragement, and support that they have shown as well as that from my sister, Leanne and her husband Dave, and my in-laws, Jim and Evelyn Roycroft, Tammy, Javier and Samantha Farias,

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

ACKNOWLEDGMENTS

Dawn, Mark and Max Hughes, and Rebecca Roycroft have helped me, my wife Bonnie and our daughter Cecelia through the many years it took to finish.

To my daughter Cecelia whom I love more than she will ever understand, perhaps until she has a child of her own, I thank first just for being her wonderful self. Secondly, she has given me perspective into what is important in life and motivation to finish as quickly as possible.

Finally, and most of all I thank my wife, Bonnie for her love, support, and encouragement. Words cannot express the debt of gratitude I owe her in this regard. She has been a fantastic mother to our daughter, putting her career on hold to stay at home with her. Cecelia's well-adjusted personality is a tribute to the care Bon provides. I also need to thank her for her infinite patience and understanding during my time at school. I could not have completed the dissertation without Bon's support. I love her always. Now we can get on with our lives.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

For Bon and Cecelia

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Table of Contents

Acknowledgements iii

List of Tables xvii

List of Figures xx

Abstract xxi

1 Introduction 1

1.1 Problem Statement and Motivation 1

1.1.1 Motivation 2

1.1.2 Problem Statement 3

1.1.2.1 Lack of Exemplars for Objects-First CS1 3

1.1.2.2 Predictors of Success for Objects-First CS1 4

1.2 Contributions & Significance of the Dissertation 5

1.2.1 Framework for Teaching the Objects-First CS1 5

1.2.2 Impact of Objects-First on CS1 7

1.2.3 A Model for Experimental Study of Curricular Change 8

1.2.4 The First Study for Objects-First 8

1.3 Chapter Outline 8

2 Predictors of Success Background 10

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

2.1 Mathematical Predictors 11

2.2 Prior Programming Experience 13

2.3 Other Cognitive, Psychological, and Behavioral Factors 14

2.3.1 Abstract Reasoning Ability 14

2.3.2 Psychological Factors 15

2.3.3 Additional Academic Predictors 17

2.3.4 Behavioral Predictors 18

2.4 Summary 19

3 The Graphical Design-Centric Objects-First CS1 20

3.1.1 OO Relationships between Classes 22

3.2 Teaching by Example 23

3.2.1 Best Little Pizza House (Nguyen, 2002) 24

3.2.1.1 Concepts Covered 24

3.2.1.2 Description 24

3.2.1.3 Discussion 24

3.2.2 Object-Oriented Sorters (Wong) 26

3.2.2.1 Concepts Covered 26

3.2.2.2 Description 26

3.2.2.3 Discussion 26

3.2.3 BouncingBall 27

3.2.3.1 Concepts Covered 27

3.2.3.2 Description 27

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

3.2.3.3 Discussion 28

3.2.4 FourSquares, TwentySquares, and HundredSquares 28

3.2.4.1 Concepts Covered 28

3.2.4.2 Description 29

3.2.4.3 Discussion 29

3.2.5 SingleColorSquares 30

3.2.5.1 Concepts Covered 30

3.2.5.2 Description 30

3.2.5.3 Discussion 31

3.2.6 User-driven SingleColorSquares 33

3.2.6.1 Concepts Covered 33

3.2.6.2 Description 33

3.2.6.3 Discussion 33

3.2.7 User-driven Rotation Control 35

3.2.7.1 Concepts Covered 35

3.2.7.2 Description 35

3.2.7.3 Discussion 35

3.2.8 AlienFace 37

3.2.8.1 Concepts Covered 37

3.2.8.2 Description 37

3.2.8.3 Discussion 37

3.2.9 Fraction 38

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

3.2.9.1 Concepts Covered 38
3.2.9.2 Description 38
3.2.9.3 Discussion 39
3.2.10 Counter 41
3.2.10.1 Concepts Covered 41
3.2.10.2 Description 41
3.2.10.3 Discussion 43
3.2.11 Grade 43
3.2.11.1 Concepts Covered 43
3.2.11.2 Description 43
3.2.11.3 Discussion 44
3.2.12 Password 44
3.2.12.1 Concepts Covered 44
3.2.12.2 Description 44
3.2.12.3 Discussion 44
3.2.13 Turtle 45
3.2.13.1 Concepts Covered 45
3.2.13.2 Description 45
3.2.13.3 Discussion 46
3.2.14 Bouncing 47
3.2.14.1 Concepts Covered 47
3.2.14.2 Description 47

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

3.2.14.3 Discussion 47
3.2.15 TicTacToe 49
3.2.15.1 Concepts Covered 49
3.2.15.2 Description 49
3.2.15.3 Discussion 49
3.3 Conclusion 53
4 Experimental Method 54
4.1 Research Questions 54
4.1.1 Traditional Predictors 55
4.1.2 Success 55
4.1.3 Resign rates 55
4.2 Subjects 56
4.3 Materials 57
4.3.1 Cornell Critical Thinking Test 57
4.3.2 Computer Programming Self-Efficacy Scale 57
4.4 Procedure 58
4.4.1 Paper and Electronic Testing 58
4.4.2 Grades 59
4.4.3 InfoSource Data Collection 60
4.4.4 Last Log Mining 60
4.4.5 Office Hour Tracker 61
4.4.6 Analyses 62

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

4.5 Variables 62

4.5.1 Independent Variables 62

4.5.2 Dependent Variables 68

5 Experimental Results 69

5.1 Success 70

5.1.1 Prior Programming Experience 70

5.1.1.1 Programmers vs. Non-programmers 71

5.1.1.2 Experience with OO Languages 71

5.1.1.3 Self-Efficacy for Object-Orientated Concepts 73

5.1.2 By Gender 74

5.1.3 By Year in School 75

5.1.4 By Major 77

5.1.5 By Reason for CS Major 80

5.1.6 Stepwise Multiple Linear Regression 82

5.1.6.1 Course Average 82

5.1.6.2 Lab Average 91

5.1.6.3 Exam Average 94

5.2 Resignations 97

5.2.1 By Gender 97

5.2.2 By Credit Hours 98

5.2.3 By Year in School 98

5.2.4 By Major 100

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

5.3 Summary 106
6 Discussion of Experimental Results 108
6.1 Lack of Gender Bias for Success and Resignations 108
6.2 Success 109
6.2.1 Prior Programming Experience 109
6.2.2 Year in School 111
6.2.3 Major/Intended Major 112
6.2.4 Reason for CS Major/Intended Major 113
6.2.5 Multivariate Models of Course Average 113
6.2.5.1 Full Models of Course Average 114
6.2.5.2 Non-Yoked Model of Course Average 115
6.2.6 Multivariate Model of Lab Average 116
6.2.7 Multivariate Model of Exam Average 116
6.2.8 Summary of Multivariate Models of Success 116
6.3 Resignations 117
6.3.1 Credit Hours 118
6.3.2 Year in School 118
6.3.3 Major/Intended Major 119
6.4 Summary and Conclusions 122
6.4.1 How do predictors of success differ for OF CS1? 122
6.4.2 A CS1 for everyone 123
7 Conclusion 126

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

8 Future Work 129
8.1 Predictors of Success and CS2 129
8.2 CS1 Not Just for CS Students 130
8.3 Leading a Horse to Water and the Lost 131
8.4 Multi-Institutional Analysis 132
8.5 Long-term 133
8.5.1 Preliminary Investigation 133
8.5.1.1 Method 133
8.5.1.2 Results 134
8.5.1.3 Discussion 135
Bibiliography 137
Appendix A 144
Appendix B 146
Appendix C 180
Appendix D 202
Appendix E 224

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

List of Tables

Table 1.1 Order of Topics in Current CS1 Java Textbooks 3

Table 3.1 Relations of the Simplified Class Diagrams 22

Table 3.2 Ordered Topic List for the Graphical Design-Centric Objects-First CS1 23

Table 5.1 Mann-Whitney U for Success by Prior Programming Experience 71

Table 5.2 t-tests for Success by Prior C++ Experience 72

Table 5.3 t-tests for Success by Prior Java Experience 73

Table 5.4 Pearson Correlations for OOP Self-Efficacy and Success 74

Table 5.5 t-test for Success by Gender 74

Table 5.6 Mean Ranks for Success by Year in School 76

Table 5.7 Kruskal-Wallis for Success by Year in School 76

Table 5.8 Mann-Whitney U for Lab Average between Seniors and Sophomores 76

Table 5.9 Major Abbreviations 77

Table 5.10 Major to Major (by type) mapping 79

Table 5.11 Mean Ranks for Success by Type of Major/Intended Major 80

Table 5.12 Kruskal-Wallis for Success by Type of Major/Intended Major 80

Table 5.13 Mean Ranks for Success by Reason for Computer Science Major/Intended Major 81

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Table 5.14 Kruskal-Wallis for Success by Reason for Computer Science Major/Intended Major 82

Table 5.15 Descriptive Statistics for Independent and Dependent Variables of Stepwise Multiple Linear Regression of Course Average 84

Table 5.16 Stepwise Multiple Linear Regression Models for Course Average 85

Table 5.17 Coefficients for Stepwise Multiple Linear Regression Models of Course Average 89

Table5. 5.18 Stepwise Linear Regression Models of Course Average sans Yoked Variables 90

Table 5.19 Coefficients for Stepwise Multiple Linear Regression of Course Average sans Yoked Variables 90

Table 5.20 Descriptive Statistics for Stepwise Multiple Linear Regression for Lab Average 92

Table 5.21 Stepwise Multiple Linear Regression for Lab Average 92

Table 5.22 Coefficients for Independent and Dependent Variables of Stepwise Multiple Linear Regression Models of Lab Average 94

Table 5.23 Descriptive Statistics for Independent and Dependent Variables of Stepwise Multiple Linear Regression for Exam Average 95

Table 5.24 Stepwise Multiple Linear Regression Models for Exam Average 96

Table 5.25 Coefficients for Stepwise Multiple Linear Regression Models of Exam Average 97

Table 5.26 Gender by Resign Rate 97

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Table 5.27 Mann-Whitney U test for Number of Credit Hours by Resign 98

Table 5.28 Year in School by Resign Rate 100

Table 5.29 Major by Resign Rate.. 103

Table 5.30 Type of Major/Intended Major by Resign Rate 105

Table 8.1 OS Course Grade by CS1 Type 134

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

List of Figures

Figure 3.1 Class Diagram for Best Little Pizza House 25

Figure 3.2 First Code Example 27

Figure 3.3 Class Diagram for Four Squares Example Program 30

Figure 3.4 Naive Design of SingleColorSquares 31

Figure 3.5 Final Design of SingleColorSquares 32

Figure 3.6 A CSE115.Dialogs.ColorDialog Object 34

Figure 3.7 Class Diagram for User-Driven Rotation Control 36

Figure 3.8 Alien Face 37

Figure 3.9 A Fraction Class 39

Figure 3.10 Fraction Class Code 40

Figure 3.11 Code for Demonstrating Round-Off Error 42

Figure 3.12 Password Validation Code 45

Figure 3.13 Code for Bouncing Balls 48

Figure 3.14 Position Class Code 51

Figure 3.15 TicTacToeBoard HashMap-based Implementation 53

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Abstract

Object-oriented programming (OOP) has made tremendous gains since its birth in the early 1960s. Since the beginning of the 1990s, many have written about the importance of introducing OOP in CS curricula. With the advent of Java, there has been considerable discussion of how to properly teach Java in introductory courses. In short, the discussion has largely dealt with the introduction of the object-oriented paradigm in CS1. Finally, in 2001, the IEEE/ACM Joint Task Force on Computing Curricula legitimized the teaching of object-oriented programming in the introductory course sequence, termed *objects-first*. No empirical investigations of the objects-first approach have appeared to date. This dissertation discusses the results of a systematic investigation of the objects-first CS1 course developed by the author. Further, it includes a model syllabus with class-tested examples for teaching object-oriented concepts in CS1.

The dissertation has examined predictors of success for the objects-first course and compares results with traditional *imperative-first* approaches. The predictors include, prior programming experience, mathematical ability, academic and psychological variables, gender, and measures of student effort. The findings show a radical difference between the predictors of success of the objects-first approach versus the imperative-first

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

approach. Most surprising is the finding that prior programming experience is not a predictor of success. Further, cognitive and academic factors such as SAT scores and critical thinking ability offer little predictive value when compared to the other predictors of success. Student effort and comfort level were found to be the strongest predictors of success.

The dissertation reveals that the objects-first CS1 is an attractive option both for large universities and liberal arts settings as well as in the AP CS curriculum. The discussion of in-class examples serves to aid instructors new to objects-first, in teaching an objects-first CS1, for which there is a lack of pedagogical examples.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Chapter 1

Introduction

1.1 Problem Statement and Motivation

The focus of this dissertation is an empirical investigation of a new approach, *objects-first* that is defined by CC2001, for the first course in computer science, CS1. A curriculum based on this approach is outlined in detail. According to the IEEE/ACM Joint Task Force on Computing Curricula's Computing Curricula 2001 Computer Science (CC2001), the objects-first approach "emphasizes the principles of object-oriented programming and design from the very beginning. … The first course … begins immediately with the notions of objects and inheritance … the course then goes on to introduce more traditional control structures, but always in the context of an overarching focus on object-oriented design." However, CC2001 simply provides a topic list for all courses including the objects-first approach. This dissertation addresses both the problem of how to teach the objects-first CS1 as well as the predictors of success for such a

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

course. This dissertation will discuss one possible implementation of such a course. The work will also examine predictors of success when this new approach is used. The thesis is that traditional predictors such as mathematical ability and prior programming experience do not hold for an objects-first approach, but that effort and comfort level do.

1.1.1 Motivation

Object-oriented programming (OOP) has made tremendous gains since its inception by Ole-Johan Dahl and Kristen Nygaard in the 1960s (see Dahl & Nygaard for a discussion). They write, "[O]bject-oriented programming is today (in the late 1990s) becoming the dominant style for implementing complex programs with large numbers of interacting components." Indeed one need only look to modern object-oriented languages such as C++, Java, and Microsoft's C# as proof of OOP's success. Since the mid-1990s, there has been considerable attention given to teaching object-oriented programming early (Perry, 1996; Woodman, Davis, & Holland, 1996). The fact that Java is closer to a pure object-oriented language than C++ has meant that educators wishing to adopt Java as a first language have needed to rethink their curricula (Bergin, Koffman, Proulx, Rasala, & Wolz, 1999; Culwin, 1999; Kölling & Rosenberg, 2001; Mitchell, 2001; Weber-Wulff, 2000).

In December 2001, the IEEE/ACM Joint Task Force on Computing Curricula finalized the model computing curricula (Curricula, 2001). In it, they have legitimized teaching OOP in the CS1 and CS2 introductory course sequence. They have set forth

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

recommendations for an objects-first approach to CS1 and CS2 that includes both OOP and object-oriented design (OOD).

In short, empirical investigation of the objects-first approach is timely and appropriate.

1.1.2 Problem Statement

1.1.2.1 Lack of Exemplars for Objects-First CS1

CC2001 helps to address the complaint of Lewis (2000) that objects-first is poorly defined (see quote in §1.2.1). There does, however, seem to be some confusion about the meaning of the term "objects-first". The handful of current CS1 book titles that claim to be objects-first do not pass muster according to the CC2001 definition. Primarily they fall short in terms of introducing control structures prior to inheritance and polymorphism (see Table 1.1).

Book	Chapter where topic is introduced				
	Objects	Selection	Iteration	Inheritance	Polymorphism
Barnes & Kölling (2003)	1	2	4	8	8
Gittleman (2002)	2	3	3	9	9
Holmes & Joyce (2001)	2	4	5	6	6
Nino & Hosch (2002)	2	6	12	14	14
Riley (2002)	1	7	10	8	9
Wu (2001)	4	6	7	14	14

Table 1.1 Order of Topics in Current CS1 Java Textbooks

More severely, these books largely give lip service to use of objects early on, after which OOP is practically abandoned. Morelli (2003) is the most severe example of this,

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

where despite "Object-Oriented Problem Solving" being in the title, the book omits chapters on inheritance and polymorphism. Further, design of solutions that make use of fundamental relationships such as composition and association are notably absent. The lack of discussion of basic OO relationships is also true of Gittleman (2002), Holmes & Joyce (2001), and Wu (2001). One might argue that these books are objects-early rather than objects-first.

Assuming one wishes to adopt an objects-first curriculum, the real problem is the dearth of exemplars for an objects-first curriculum. Teaching an imperatives-first CS1 is easy (relatively speaking) since there are a wealth of textbooks, sample syllabi, and examples used to motivate the concepts. Those adopting an objects-first approach are left largely to innovate curricula on their own. The research question becomes, "How does one teach an objects-first CS1 in a motivating manner that can be understood by one's students?" This dissertation answers this question by providing a model curriculum and syllabus with detailed explanation of in-class examples.

It should be noted that this dissertation does not assess learning outcomes for the course. The problem of assessing CS1 knowledge as well as object-oriented concepts is an open research question.

1.1.2.2 Predictors of Success for Objects-First CS1

Given the trend towards teaching CS1 & CS2 using an objects-first approach, the question is, "How does the objects-first approach change the nature of what is currently

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

understood about introductory courses?" This research seeks to investigate the relationship of the objects-first approach to *predictors of success* for CS1. Predictors of success are factors that can be studied to ascertain how well a student will perform in either the course as a whole or in particular parts of a course. For instance, mathematical ability is traditionally viewed as a predictor of success for study in computer science. Included are both factors the students enter the class with, such as mathematical ability or prior programming experience, and behaviors of the students during the class, such as attendance and hours worked at a job.

1.2 Contributions & Significance of the Dissertation

The proposed research will have contributions to the field of computer science in the following ways.

1.2.1 Framework for Teaching the Objects-First CS1

Lewis (2000) argues that the term "objects-first", and its sister concept "objects-early," are not well defined. He writes, "No matter what your definition of objects first is, it is likely to be different from that of the person next to you. In papers, presentations, textbooks, and even hallway conversations, we should always clarify this term to ensure clear communication" (p. 247). This research takes up Lewis's charge and will contribute to the definition of objects-first, providing a framework for teaching an objects-first CS1.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

The framework also addresses issues raised in (Curricula, 2001), such as:

> Programming courses often focus on syntax and the particular characteristics of a programming language, leading students to concentrate on these relatively unimportant details rather than the underlying algorithmic skills. This focus on details means that many students fail to comprehend the essential algorithmic model that transcends particular programming languages. Moreover, concentrating on the mechanistic details of programming constructs often leaves students to figure out the essential character of programming through an ad hoc process of trial and error. Such courses thus risk leaving students who are at the very beginning of their academic careers to flounder on their own with respect to the complex activity of programming.
>
> Introductory programming courses often oversimplify the programming process to make it accessible to beginning students, giving too little weight to design, analysis, and testing relative to the conceptually simpler process of coding. Thus, the superficial impression students take from their mastery of programming skills masks fundamental shortcomings that will limit their ability to adapt to different kinds of problems and problem-solving contexts in the future. (pp. 23-24)

The approach that has been developed has a greater focus on the design and analysis components than mere programming proficiency.

The early introduction of design patterns (Gamma, Helm, Johnson, & Vlissides, 1995) not only addresses the previous concerns, but also answers the call for a more central role for design patterns (Astrachan, Mitchener, Berry, & Cox, 1998; Curricula, 2001; Nguyen & Wong, 2001; Walingford, 1996).

Finally the framework also contains motivation in the way of examples for various OO and object-oriented design (OOD) concepts.

The framework can likewise serve as a guide for secondary schools, given the recent move of the Advanced Placement Computer Science program adopting Java as the language of choice.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

1.2.2 Impact of Objects-First on CS1

As outlined in the problem statement, the object-oriented paradigm is more than just a fad. It provides substantial expressive and organizational power, as is evidenced by the widespread use of object-oriented languages such as C++ and Java. The non-trivial nature of the paradigm is echoed by the recommendations of CC2001 in not only including object-oriented programming in the curricula, but advocating its use in the introductory sequence.

This dissertation represents a first step in understanding the ramifications of the use of the objects-first approach in CS1 with regard to predictors of success. Additionally, the current research contradicts the conjecture of CC2001 that "[p]rogramming-intensive courses disadvantage students who have no prior exposure to computers .... As a result, students who are new to computing are often overwhelmed." That is, as will be shown, prior programming experience is not a predictor of success for the objects-first CS1.

The findings of this dissertation dispute the widely-held belief, supported by past literature, that mathematical ability is a predictor of success for CS1 courses. The research found a negligible predictive value for SAT math scores. The predictive value of SAT math scores for the objects-first CS1 is lower than the predictive value of SAT scores in general for freshman performance.

This investigation opens the door for additional research into the overall pedagogical advantages of the objects-first approach.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

1.2.3 A Model for Experimental Study of Curricular Change

Unfortunately, much of current discussions of innovations in CS1 curricula are limited to experience papers. While dissemination of new techniques for teaching is important, these papers lack any experimental evidence for the effect of, or advantage in using the new techniques. Without such experimental work, debates about curricular change become as meaningful as coffee-drinkers arguing over how to take one's coffee.

This dissertation serves as a model for those individuals wishing to investigate predictors of success for their own CS1 courses. The model of such investigation encourages researchers not to simply accept the obvious or long-held beliefs. For instance, unlike past research into predictors of success for CS1, this dissertation also includes several measures of student effort.

1.2.4 The First Study for Objects-First

Currently there are no published data investigating the use of objects-first in the introductory curriculum. While some research exists for other types of approaches, mainly imperatives-first (see Chapter 2), the objects-first investigations are non-existent. This dissertation represents the first empirical study on the use of objects-first in CS1.

1.3 Chapter Outline

The remainder of the dissertation is organized as follows. Chapter 2 discusses the past research into predictors of success for imperatives-first CS1 courses. Prior

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

programming experience and mathematics are shown to be predictive of success for these courses. Behavioral, psychological, and academic predictors are also discussed.

Chapter 3 provides the model curriculum developed to answer the problem of how one teaches an objects-first CS1. The chapter includes a sample syllabus (ordered topic list) as well as the full set of in-class examples used to teach the objects-first CS1.

Chapter 4 introduces the experimental methodology employed in this dissertation to identify predictors of success for the objects-first CS1. Research questions based on the imperatives-first literature are formalized. The independent variables are identified and defined. The collection process and tests used to gather the data are explained. Finally, the different measures of success are defined.

Chapter 5 details the statistical tests used to answer the research questions and identify predictors of success for the objects-first CS1. Chapter 6 summarizes and discusses the results presented in Chapter 5.

Chapter 7 provides the conclusions of this dissertation. Chapter 8 discusses the future research and open questions resulting from this dissertation.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Chapter 2

Predictors of Success Background

This chapter examines previous attempts to identify predictors of success for non-objects-first approaches and related research. The discussion includes traditional predictors such as mathematical ability (§2.1) and prior programming experience (§2.2), as well as cognitive factors such as abstract reasoning (§2.3.1), personality traits (§2.3.2), academic (§2.3.3) and behavioral (§2.3.4) variables. The independent variables (predictors) are measured against various measures of the dependent variables (success), such as overall course grade, programming assignment performance, exam performance, and in some studies, homework assignments.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

2.1 Mathematical Predictors

The prior research involving mathematics as a predictor of success includes prior math course exposure and performance in those courses, mathematical ability and achievement, as well as, mathematical aptitude, for instance SAT math scores.

Mathematical ability is often cited as a predictor of success in computer science. Leeper and Silver (1982) included SAT math scores in their analysis. Indeed they found SAT math scores to have the second strongest correlation with course grade, $r = .3777$ (SAT verbal scores were the strongest predictor). That is, SAT math scores accounted for 14.27% of the variance in course grades. While Leeper and Silver did not report the statistical significance of these values, the linear regression model based on these and other factors was significant at the .05 level. In other words, there is 95% certainty that the results did not occur due to chance. This dissertation examines whether such a relationship exists for the objects-first approach.

Nowaczyk's (1983) investigation strengthened the argument for mathematical ability as a predictor of success. In particular, he measured performance in prior math courses. Coupled with prior English course performance, the measures accounted for a statistically significant amount of variance in course grade for an introductory COBOL programming course. While success in prior math courses may be a helpful measure, self-report data is a notoriously imperfect and perhaps inflated measure. For that reason, the dissertation does not include such data.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Evans and Simkin (1989) examined the effect of the number of high school math classes a student took, rather than grades in math classes. They found the number of high school math classes to be the most significant predictor for cumulative homework score in an introductory BASIC programming course.

Byrnes and Lyons (2001) also noted the correlation between mathematical ability and course grade in an introductory BASIC programming course. Through the use of students' scores on the Irish Leaving Certificate examination the researchers assessed mathematical achievement. The Irish Leaving Certificate exam contains six subjects including math and is given as a pre-college entrance examination, in the same way the SAT is used in the United States. Their analysis revealed a statistically significant correlation, $r = .353$, $p < .01$. While significant, the predictive value is still quite low, $r^2 = .125$, i.e. the amount of variance accounted for is only 12.5%. In other words, knowing a student's math score will lead to predictions about course grade that are only 12.5% more accurate.

Cantwell Wilson and Shrock's (2001) investigation revealed the number of semesters of high school math as the second most important factor in predicting student course grades in an introductory computer science course. Our preliminary investigations into the role of number of high school math courses have shown that this relationship does not hold for the objects-first approach.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

In summary, we note that the role of mathematical ability as a predictor of success has been well studied for non-objects-first approaches. These data will be used as a reference point for the experimental work of this dissertation.

2.2 Prior Programming Experience

The notion of prior programming experience as a predictor of success seems obvious. Evans and Simkin noted that prior BASIC programming experience was a predictor for scores on the second exam in an introductory BASIC programming course.

Hagan and Markham (2000) also examined the effect of prior programming experience on performance in an introductory Java programming course. They found that the students with prior programming experience did better on the first and last stages of the course project (the second stage was omitted from their analysis) as well as the first two exams and the overall course grade. The differences were statistically significant for each. The final exam was unaffected by prior programming experience. An ANOVA was performed to assess the effect of number of languages known. Indeed statistical significance was identified for each of the previous dependent variables, namely first and last stages of the course project, the first two exams, and course grade. Unfortunately, Hagan and Markham's discussion does not include the statistical measures that show where the differences occurred.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Cantwell Wilson and Shrock found that having taken previous programming courses was a statistically significant predictor for midterm exams in the researchers' introductory programming course.

There are no published studies regarding this effect for the objects-first approach. The current research will investigate what effect (if any) prior programming experience has when an objects-first approach is used.

2.3 Other Cognitive, Psychological, and Behavioral Factors

This section discusses psychological, behavioral, and cognitive factors not included in the preceding review. The factors include abstract reasoning and problem-solving as well as personality traits. A discussion of relevant academic background is also included.

2.3.1 Abstract Reasoning Ability

Kurtz (1980) categorized introductory computer science students in terms of "formal (or abstract) reasoning abilities". He identified three different levels of development: *late concrete*, *early formal*, and *late formal*. His testing instrument to categorize students included questions dealing with direct and inverse proportions, probability, and propositional and deductive logic. He found that the first and last developmental levels were "strong predictors of poor and outstanding performance, respectively; and the [developmental] level predicts performance on tests better than performance on programs." Unfortunately, his classification scheme is non-standard and has not been

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

validated. Further, Kurtz provides little insight into what is meant by the three categories.

Nowaczyk (1983) added to the overall picture. He found significant positive relationships between course grade and logic problem-solving ability as well as algebraic word problem-solving ability.

Evans and Simkin found that "letter-set problem-solving" ability predicted performance on the matching portion of a BASIC programming exam. The letter-set problem-solving test involves discernment of patterns of letters. They further found that ability to draw analogies and follow directions along with spatial relations (ability to reason in 2D space) predicted final exam score.

Unfortunately many of the measures mentioned are arbitrary and have not been tested for validity or reliability. Indeed it is exceedingly difficult to find valid and reliable tests and measures for such concepts, especially those that can be administered and scored by laypersons. Additionally many of these instruments are cost-prohibitive. For that reason, these measures were omitted from this dissertation.

2.3.2 Psychological Factors

Nowaczyk included computer anxiety and *locus of control* (an individual's notion of how s/he can control what happens) in his study. However, these showed no predictive value. He explained, "very few students showed any computer anxiety and most of the students were *internalizers* (with regard to locus of control)." Internalizers are

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

individuals who tend to believe that success or failure is dependent on factors inside them as opposed to outside forces. They can be contrasted with *externalizers*, who tend to believe that success or failure is due to the general environment.

Cantwell Wilson and Shrock included a related measure, namely attribution of success or failure on the midterm exam. Students were asked whether they were happy or dissatisfied with their midterm exam scores. They were then asked to rate various reasons for success or failure on the exam. The researchers found that attribution to luck was negatively correlated with midterm score. The idea of attribution of success (related to internalization vs. externalization) is included in the dissertation.

Evans and Simkin (1989) noted the emergence of "cognitive factors as important explanatory variables." In the study, they examined the effect of Myers-Briggs personality traits along with problem-solving ability. The Myers-Briggs Type Indicator (MBTI) rates individuals on four opposing pairs of personality traits (see Appendix A). Evans and Simkin identified the MBTI trait, *sensing*, as a predictor for homework. The traits *intuition*, *thinking*, and *judging* were predictors for the first exam. The trait *introversion* was a predictor for the second exam. While the findings regarding the Myers-Briggs ratings are intriguing, it is cost prohibitive to include this measure in the present work.

Cantwell Wilson and Shrock (2001) identified a student's comfort level as the best predictor of success for course grade. This discovery was based on a linear regression model of twelve variables. The variables examined were gender, previous programming

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

experience, previous non-programming computer experience, encouragement to pursue computer science, comfort level, work style preference, attribution, self-efficacy, and math background. It should be noted that math background was second to comfort level in terms of predictive value. Comfort level is also included in this dissertation.

2.3.3 Additional Academic Predictors

Leeper and Silver's (1982) study included SAT verbal (and math), rank in high school, and units of English, mathematics, science, and foreign language completed in high school. These variables were compared to overall course grade. The most significant variable was SAT verbal score, followed by SAT math, and then number of units of English in high school. A linear regression was performed to build a predictive model. The model was able to account for 26% of the variance; i.e., it had a predictive value of 1 in 4. We will examine the effect of SAT verbal and high school rank as well.

Byrnes and Lyons (2001) found performance on the Science portion of the Irish Leaving Certificate to be highly positively correlated with performance in an introductory programming course, $r = .572, p < .01$. Irish Leaving Certificate scores for English and foreign language were not correlated with course grade. Unfortunately, this dissertation cannot take advantage of such scores since the SATs do not include a science section.

Mazlack's (1980) examination included the IBM Programmer's Aptitude Test (PAT) as a predictor in an introductory FORTRAN programming course. The PAT was

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

significantly correlated only to scores on the midterm. However, when examined against the programming parts of the course, no significant correlation could be discovered.

2.3.4 Behavioral Predictors

Cantwell Wilson and Shrock included playing of games on the computer in their study. The type of computer game play was not specified. They found, however, that game playing had a negative impact on the midterm score. The explanations for such an impact are varied. One possibility is that students who are spending considerable time playing games are neglecting their studies. Alternatively it may point at a deeper issue of lack of interest in and appreciation of what computer science really is as a field. Indeed, Sheard and Hagan's (1998) report on a group of "repeat" students of an introductory programming class found that these students had "much less desire for learning computing, and in particular programming." For many, the computing course was not their first choice. The repeat group also had greater outside work commitments, including work shifts that overlapped class time.

This dissertation also includes interest in computer science measured in terms of reason for selecting computer science as a major. Students who had jobs were asked for the average number of hours a week they worked at their jobs.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

2.4 Summary

In summary the idea of mathematics and prior programming experience as predictors of success have been well documented for non-objects-first approaches. These data will be used as comparison points for findings for an objects-first approach in this dissertation.

Other academic predictors such as SAT verbal scores, high school rank and number of English, math, science and foreign language were also investigated. SAT verbal scores yielded some promise, and are also present in the analysis.

Psychological factors were also tested with mixed results. Nowaczyk was unable to find any relationship for locus of control, although he noted his sample was lacking. Issues such as attribution to success or failure and degree of comfort with the course did in fact reveal significant relationships and will therefore be tested for the objects-first approach.

Finally, behavioral factors were also found to be important. Amount of time playing computer games negatively impacted scores. Further evidence pointed to the negative effect of too many hours working at jobs as well as lack of interest in the field.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Chapter 3

The Graphical Design-Centric Objects-First CS1

This chapter discusses the content of the graphical design-centric objects-first CS1 course that is the focus of this dissertation. The course is graphical in the sense that graphics are used from the very first example students see through the entire course. Students are taught from the beginning to use (simplified) UML class diagrams to express their OO solutions. Additionally, the topic of design patterns is included. The emphasis on class diagrams and design patterns leads to a design-centric curriculum. The course strictly adheres to CC2001's requirement for objects-first that control structures are discussed after the *Three Pillars of Object-Orientation*, namely encapsulation, inheritance, and polymorphism.

While CC2001 provides a general definition, namely, that in the OF CS1, object-oriented topics such as encapsulation, inheritance, and polymorphism are covered before

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

more traditional topics such as selection and iteration, leaves issues of content untouched for teachers of the material. The problem is one of a severe lack of examples used to teach an objects-first CS1. This chapter will serve as a guide for those instructors wishing to adopt an objects-first approach. That is, it answers the question, "How does one teach an OF CS1? And what does an OF CS1 course include?" It also provides a model syllabus (see Table 3.2) with examples for adaptation.

The intent of this chapter is not to teach object-oriented design and programming, but rather to act as a set of examples to be used by those wishing to adopt an objects-first approach. Therefore, the chapter assumes that readers are familiar with the Java programming language (Joy, Steele, Gosling, & Bracha, 2000), design patterns (Gamma et al., 1995) and basic UML class diagrams (Booch, Rumbaugh, & Jacobson, 1999).

The course was based on the CS015 course at Brown University developed by Andries van Dam (2002). The CS015 course ambitiously covers CS1 and CS2 material in a single semester. Therefore, the first step we took was to split the material in half. Additionally, the reader will note there is only a single example retained from the CS015 class. Lab assignments[1] have likewise been created anew. UB's CS1 curriculum includes a *just-in-time* treatment of design patterns not found in the CS015 course. That is, rather than discussing design patterns as a separate topic as in CS015, UB's CS1 discusses whenever them in the context of the course where they solve the problem at

[1] Lab assignments are not discussed in this text. Rather only those examples used in lecture to motivate the concepts.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

hand. As a final departure from CS015, arrays have been banished in favor of Java's collection classes.

3.1.1 OO Relationships between Classes

The graphical design-centric objects-first CS1 takes its ordering of topics not only from CC2001, but also from the basic relationships found in the Unified Modeling Language's (UML) class diagrams (Booch et al., 1999). In the course, we use a simplified version of UML class diagrams (see Table 3.1) where each relation is given a specific semantics so that code can be generated directly from the diagram. This simplification has its roots in Brown's CS015 course.

Relation	Formal Name	Colloquial Name	Description
A ◇—— B	Composition	Has a	Models the whole-part relationship where the whole (A) is responsible for the creation (and deallocation) of the part (B).
A -----> B	Dependency	Uses a	A non-instance variable of another class that represents the fact that two classes must be used together. Local variables, formal parameters, and return types for which there is no corresponding instance variable qualify. A *uses a* B.
A ——> B	Association	Knows a	Models the general case of a non-composition instance variable. A *knows a* B.
A ——▷ B	Generalization	Is a	UML representation of inheritance. A `extends` B.
A ----▷ B	Realization	Acts as	UML representation of implementing an interface. A *implements* B.

Table 3.1 Relations of the Simplified Class Diagrams

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Table 3.2 shows an ordered list of topics used in the graphical design-centric objects-first CS1 course that is the subject of investigation in this dissertation. Note the focus on the object-oriented core of material early in the semester, with control structures occurring later. Thus, the course meets the litmus test of objects-first provided by CC2001.

Topics
Abstraction & Modeling
Encapsulation, UML, Class vs. Object, Composition
Non-constructor Methods, Parameters & Return Values, Association
Inheritance, CRC cards, Abstract Classes
Polymorphism, State Design Pattern
Interfaces
OO Case Study with Graphics, Decorator Design Pattern
Arithmetic & Boolean Expressions
Selection & Iteration
Generic `ArrayList` & `HashMap`, Iterators

Table 3.2 Ordered Topic List for the Graphical Design-Centric Objects-First CS1

3.2 Teaching by Example

The Graphical Design-Centric Objects-First CS1 at University at Buffalo, SUNY, has been taught through the use of examples. The goal is to use examples that are interesting, compelling, and understandable to aid students' understanding of design and implementation choices. Each example in this section starts with the concepts covered for that example. The design and programming examples generally start with a problem specification as given in the Description sections below. Finally, each example contains

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

a Discussion section that highlights the key points of the example in question, including relevant class diagrams, code snippets, and screenshots.

3.2.1 Best Little Pizza House (Nguyen, 2002)

3.2.1.1 Concepts Covered

Abstraction, modeling, correctness, robustness, reusability, and extensibility.

3.2.1.2 Description

Campus Pizza is having a special today on two different pizzas. The pizzas have the same toppings. One is round, sells for $3.99, and is 5" in diameter. The other is a 4" x 6" rectangular pizza that sells for $4.99. Which is the better deal?

3.2.1.3 Discussion

First, students are led through the solution to this problem, namely, comparing the price per area for each pizza. They are asked to consider writing a program to solve this problem and then asked if it is a *good* program; i.e., would they buy it? Students quickly point out that there are problems if the pizza place changes the price or size, at which point the following formulae are proposed:

1) *price per square inch of pizza 1* = *price of pizza 1* / (π * (*radius of pizza 1*)^2)

2) *price per square inch of pizza 2* = *price of pizza 2* / ((*width of pizza 2*) * (*length of pizza 2*))

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

The students then complain of having two pizzas of the same shape to compare. They also note the possibility of new shapes. Therefore an OO approach is suggested. The students are walked through the idea of mining the problem domain for nouns. A diagram similar to that of Figure 3.1 is drawn to illustrate the structure of the problem and its solution.

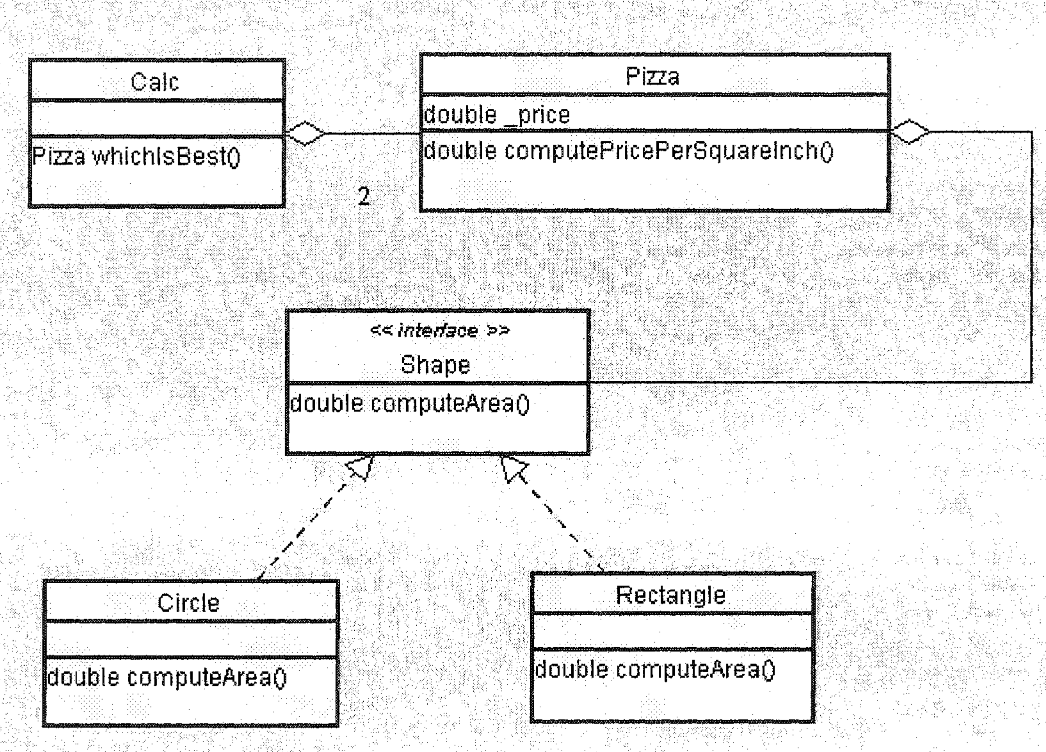


Figure 3.1 Class Diagram for Best Little Pizza House

As the diagram shows, the pizza calculator needs to be able to tell us which of the 2 pizzas is the better deal. A pizza has a price and a shape and it can use that information to compute its price per square inch. Every shape knows how to compute its own area. After seeing this diagram, the students are largely satisfied with the solution.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

3.2.2 Object-Oriented Sorters (Wong)

3.2.2.1 Concepts Covered

OO vs. imperative, smart objects vs. dumb data, introduction to encapsulation.

3.2.2.2 Description

Markers are passed out to a few students who are asked to write a number on a sheet of paper. The students then stand with their numbers facing the class. The task is to sort the numbers.

3.2.2.3 Discussion

The first attempt uses selection sort to show a process working on "dumb" data. A discussion is begun about the fact that selection sort fails to take advantage of the abilities of the students, that is, college students know how to put numbers in order.

A new group of students is then asked to repeat the process of writing numbers on paper and stand facing the class. Then these students are told to put themselves in order.

A discussion follows that points out that these students are "smart," with abilities (just like objects), and that the work is done through student communication (albeit nonverbal communication). The metaphor is explained as a lead-in to OO ideas.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

3.2.3 BouncingBall

3.2.3.1 Concepts Covered

Introduction to Java source code, editing and compiling code, and running applets.

3.2.3.2 Description

The first code sample that is given in class is a Java applet that creates an instance of a class named `BouncingBall` (see Figure 3.2). When the program is run, the students see a red "ball" bouncing off the sides, top, and bottom of the screen.

```
package BounceDemo;
/**
 * MyApplet.java
 *
 *
 * Created: Tue Sep 11 20:56:32 2001
 *
 * @author Philip R Ventura
 */

public class MyApplet extends cs015.SP.Applet{

  // declare the ball as an instance variable of MyApp
  private Demos.Bounce.BouncingBall _ball;

  public MyApplet (){
    _ball = new  Demos.Bounce.BouncingBall();
 }
}// MyApplet
```

Figure 3.2 First Code Example

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

3.2.3.3 Discussion

This simple example shows that despite the fact that none of the students have the required Java knowledge to build the bouncing ball, they can still make use of the bouncing ball class.

Furthermore, it represents a significant departure from the first code examples given in most texts and courses. In the worse of cases, the first example students see is "Hello, World!" While attempts have been made to make it more object-oriented (Weisert, 2002; Westfall, 2001; Xing & Belkhouche, 2003), the bottom line is that a) it is not interesting for students and b) provides very little in the foundations of OO to build upon.

Usually the first example of a class that students see contains only primitive data. While it is arguably better than the original "Hello, World!" it still falls short of the basic idea of objects being composed of other objects. In the bouncing ball example, students see that objects have relationships with other objects and that this idea is both primary and fundamental.

3.2.4 FourSquares, TwentySquares, and HundredSquares

3.2.4.1 Concepts Covered

Objects vs. classes, more composition, manageable complexity, top-level classes, design for re-use, and introduction to delegation.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

3.2.4.2 Description

Create a program that shows four squares on the screen. Expand it to have twenty squares, then one hundred squares.

3.2.4.3 Discussion

First the students are shown the code to create a single instance of the `RandomSquare` class in the applet. This serves as a review of the first code example they saw. The program is compiled and run. The students are asked to note the properties of the `RandomSquare` class. Invariably they note all the properties: size, position, color, and speed of rotation.

Three more squares are added and the program is run again. To address the issue of twenty squares, a discussion about the number of lines of code necessary to tackle such a task is undertaken. As a manner of reducing this complexity students are asked to consider ways of reusing code that was already written. Of course, creating multiple instance of the applet would be problematic, leading to multiple windows each with four squares. The concept of a top-level class, that is, a class that represent the workings of the program at hand is introduced. This leads to the design seen in Figure 3.3. Here the code for creation of the squares is moved into the `FourSquares` class.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

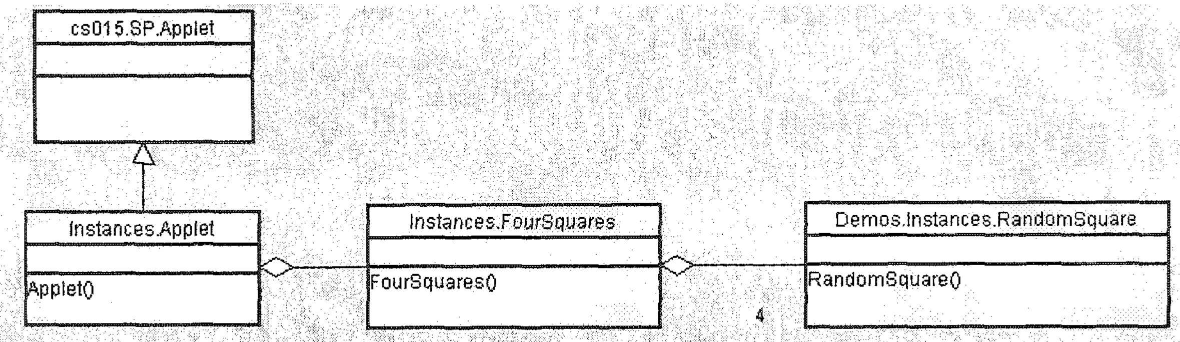


Figure 3.3 Class Diagram for Four Squares Example Program

Now students are asked how many `FourSquares` they need to make up twenty squares. The `TwentySquares` class is created which has five `FourSquares`. The program is run again and the exercise repeated to create `HundredSquares`.

The fact that the code for each class is quite small is brought to the students' attention. Furthermore, the composition of classes, delegation, reuse of code, and iterative design are all teased out.

3.2.5 SingleColorSquares

3.2.5.1 Concepts Covered

Non-constructor methods, method calls both with and without parameters, writing methods that take parameters, and delegation, again.

3.2.5.2 Description

1) Write a program that displays four random squares on the screen that are all the same color.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

2) Change that program so that they are all the same random color.

3.2.5.3 Discussion

First, students are shown how to modify the design and code from Section 3.2.3.3 to have four single colored squares. Initially students want to modify the constructor of the `FourSquares` class to make all four squares the same color. However, it is pointed out that doing so changes what `FourSquares` models and that other code may already rely on the current model. Then, the attempt is made to create a `SingleColorSquares` class that has four `RandomSquares` (see Figure 3.4). As the class diagram shows, there is no structural difference between the two classes.

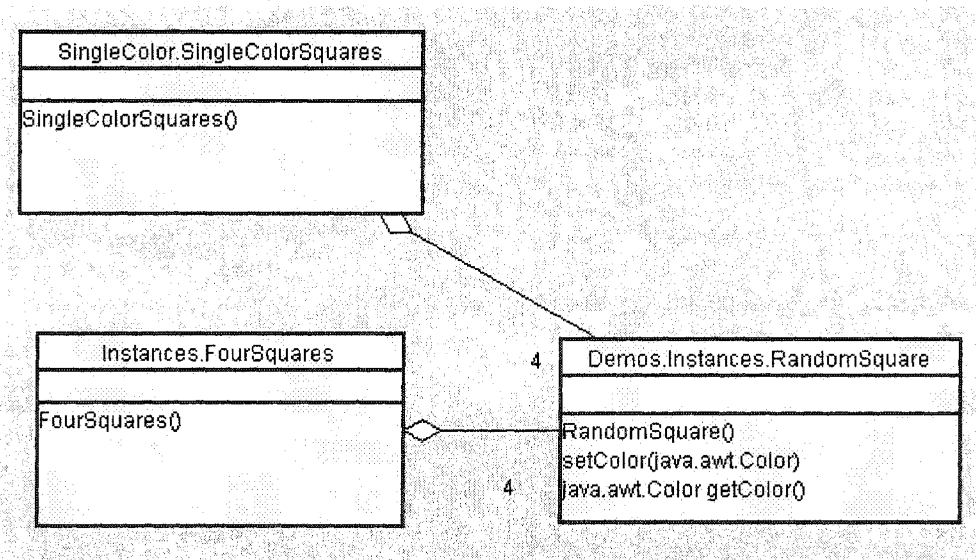


Figure 3.4 Naive Design of SingleColorSquares

Therefore, `SingleColorSquares` can be built upon `FourSquares` (see Figure 3.5). The composition relation is utilized making `FourSquares` a sub-part of `SingleColorSquares`.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

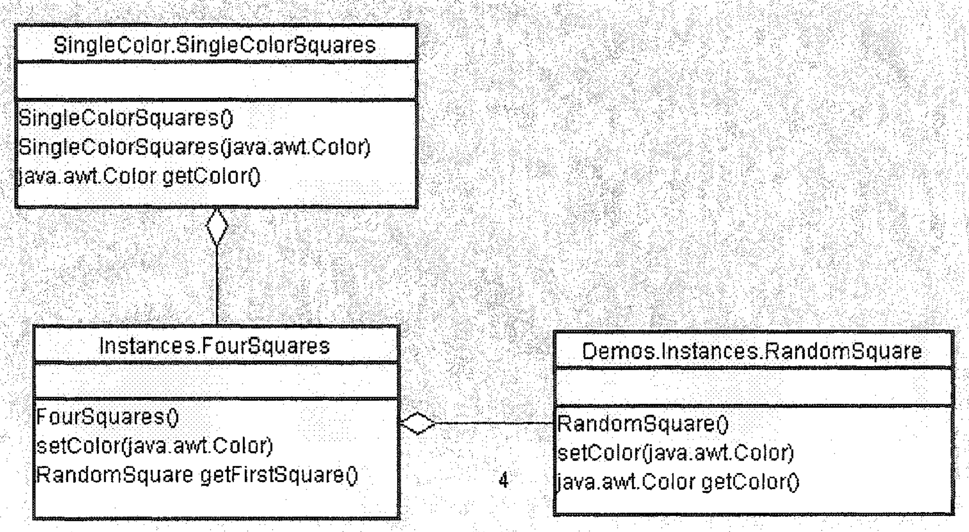


Figure 3.5 Final Design of SingleColorSquares

Once the composition relation is used, the need for the `setColor` method in `FourSquares` becomes apparent. Without it, there is no way to change the color of the squares from within `SingleColorSquares`.

To address the requirements of the second part of the problem, namely to have `SingleColorSquares` be a random color, the `getFirstSquare` method is added to `FourSquares`. This allows the `SingleColorSquares` constructor to simply make all the squares the same color as the first square.

Finally, the `getColor` method of `SingleColorSquares` is written. The first attempt requires `setColor` to store the current color as a property of the `SingleColorSquares` class. Afterwards, delegation is used to remove the need for the property. Instead, the code for `getColor` in `SingleColorSquares` simply returns the color of the first square as obtained by the `getFirstSquare` method in

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

FourSquares. This last point helps illustrate that delegation alleviates responsibility. It also provides an opportunity to discuss class invariants. When SingleColorSquares has the color property, the invariant is that the color property is the same as the color of the squares.

3.2.6 User-driven SingleColorSquares

3.2.6.1 Concepts Covered

Inheritance, and Event-driven programming.

3.2.6.2 Description

Students are asked to create a program that allows the user to change the color of four RandomSquare objects that have the same color.

3.2.6.3 Discussion

After explaining the concept of inheritance and stressing the idea that subclasses are specializations of their respective super classes, students are asked to reconsider the design of SingleColorSquares (refer back to Figure 3.5). Students are asked to consider what SingleColorSquares really is, namely a special type of FourSquares. Therefore, a new version of SingleColorSquares is created that extends FourSquares. After showing how to write the code to extend a class, the program is run creating an instance of the new SingleColorSquares class and

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

showing that four squares appear on the screen. A discussion ensues about the constructor chaining that results in the appearance of the four squares. The new `SingleColorSquares` class is then modified to ensure that all four squares are the same color.

Students are then told of the `CSE115.Dialogs.ColorDialog` class and its `approvePressed` method. The class shows model a fairly intricate color dialog (see Figure 3.6) that after the user selects a color and presses the OK button, calls its own `approvePressed` method.

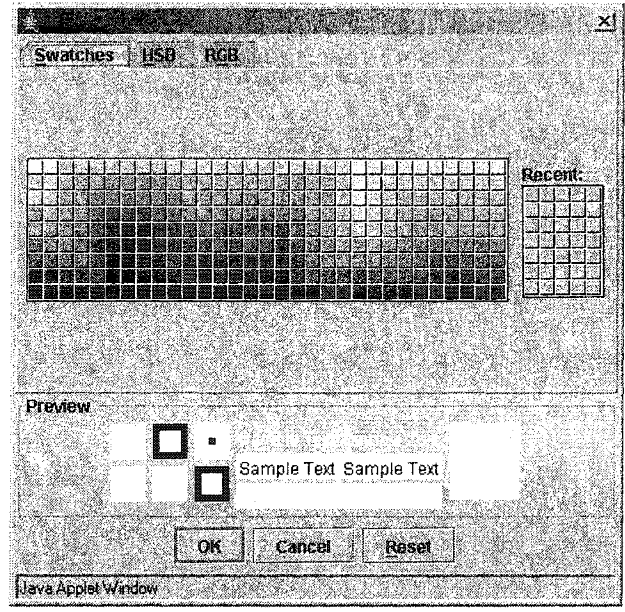


Figure 3.6 A CSE115.Dialogs.ColorDialog Object

This sets the stage for the introduction of method overriding, which is used to change the color of the squares.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

3.2.7 User-driven Rotation Control

3.2.7.1 Concepts Covered

Polymorphism and the State design pattern (Gamma et al., 1995).

3.2.7.2 Description

Students are asked to write a program that displays four random squares on the screen. There should be a rotation menu with one menu item, Stop. When Stop is selected, the squares should stop rotating and the text of the menu item should change to Start. When Start is selected, the squares should start rotating again and the text of the menu item as well as its behavior should change back to Stop.

3.2.7.3 Discussion

Prior to the introduction of the example, random students are selected and asked to give an example of an object. Students invariably name concrete objects (usually ones in the classroom). The lecture continues by telling the students that objects can be used to model intangible entities such as behaviors (for instance, the Strategy (Gamma et al., 1995) and State design patterns). Students are first led through experimentation of the use of interfaces to create a simple `MenuBehavior` that print a message to the screen. Then they are asked to modify the design and code to create the `StopBehavior` class. Finally, the State design pattern is employed to allow changes between starting/stopping

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

the rotation of the squares. Figure 3.7 shows the design of the final version of the solution.

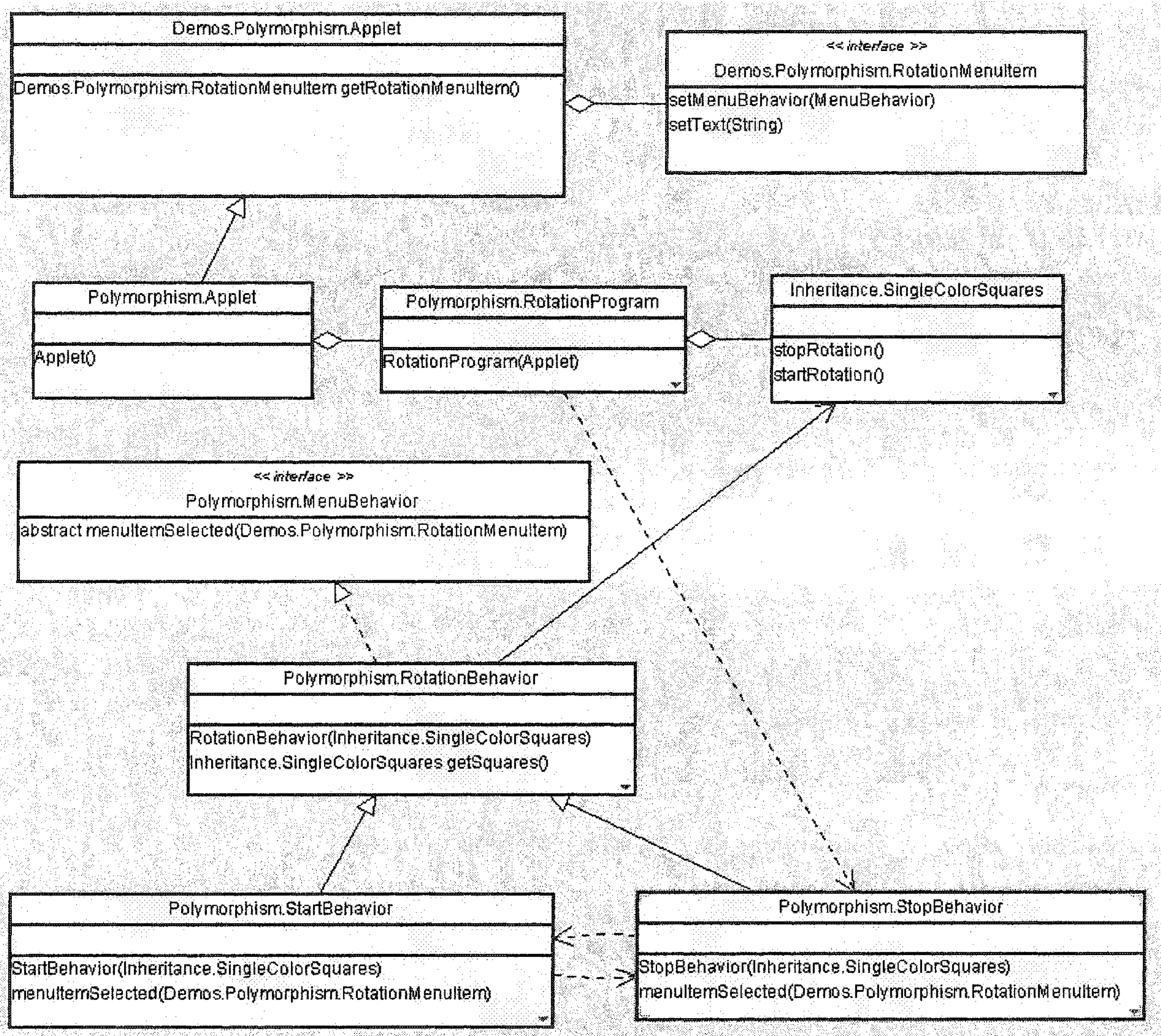


Figure 3.7 Class Diagram for User-Driven Rotation Control

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

3.2.8 AlienFace

3.2.8.1 Concepts Covered

An examination of how previous graphical examples work, review of Three Pillars of Object-Oriented Programming, the State design pattern revisited.

3.2.8.2 Description

This example was adapted from an in-class example created by Andres van Dam (2003). Students are asked to write a program that displays an alien face as in Figure 3.8 below.

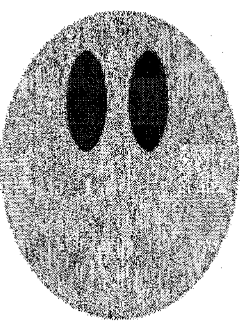


Figure 3.8 Alien Face

3.2.8.3 Discussion

The example introduces students to NGP (Chotin, 2003), a graphics library created by Matt Chotin at Brown University. All the graphical examples in class are built upon NGP. NGP provides a number of abstractions that simplify the creation of graphical user

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

interfaces, 2-D vector-based graphics, and playback of sound. NGP also serves an example of good object-oriented design.

The creation of the alien face requires the placement and sizing of the ellipses for the face and both eyes. Then students are led through an exercise of creating multiple instances of the alien face, which requires the ability to specify a location for the alien face as a whole. Students are therefore introduced to the idea of relative positioning.

Finally the movement of the alien face is animated. The students are asked to consider how movies work...that is, a picture is shown for a small amount of time and then another similar picture is displayed. This causes apparent motion. To achieve this effect programmatically, the `NGP.Timer` class is used. Timer objects call their own activate method every user-specified number of milliseconds. Finally as a review of the state design pattern and to illustrate event-driven programming, the students are asked to design and implement the code to have a push button that starts and stops the aliens movement on the screen.

3.2.9 Fraction

3.2.9.1 Concepts Covered

Basics of numerics, introduction to representation, and mixing types.

3.2.9.2 Description

Students are asked to consider the creation of a fraction class.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

3.2.9.3 Discussion

The students are asked to model fractions as a class (see Figure 3.9). First they are asked to identify the properties, namely a numerator and denominator. Then possible functionality is discussed, namely creation of a fraction where the numerator and denominator are known, creating a fraction from a whole number, and creating the fraction with value 0. Also introduced is the need to display fractions (`toString`), convert them to a real number (`eval`) and add fractions together. The code for the Fraction class appears in Figure 3.10.

Fraction.Fraction
int _numerator int _denominator
Fraction() Fraction(int num) Fraction(int num, int denom) String toString() Fraction add(Fraction rhs) double eval()

Figure 3.9 A Fraction Class

The Java type `int` is introduced along with the fact that it is a primitive type. The constructors are written (without error-checking). Then the `toString` method is written to show how to get a printable representation of an object in Java.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
package Fraction;

/**
 * Fraction.java
 * @author Philip R Ventura
 */

public class Fraction {

    private int _numerator;
    private int _denominator;

    public Fraction(int num, int denom) {
     _numerator = num;
     _denominator = denom;
    }

    public Fraction(int num) {
     _numerator = num;   _denominator = 1;
    }

    public Fraction() {
     _numerator = 0;   _denominator = 1;
    }

    public String toString() {
     return _numerator + "/" + _denominator;
    }

    public Fraction add(Fraction rhs) {
     return new Fraction(_numerator * rhs._denominator +
                  rhs._numerator * _denominator,
                  _denominator * rhs._denominator);
    }

    public double eval() {
// First try
//   return _numerator / denom;

     return (double) _numerator / _denominator;
    }
} // Fraction
```

Figure 3.10 Fraction Class Code

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

The reason for `add` only needing a single parameter is discussed (it is called a `Fraction` object). The fact that `add` is non-mutative is also pointed out. Many students are surprised to learn that the `add` method can directly access the numerator and denominator of `rhs`.

Finally `eval` is written. The first attempt at the implementation of `return _numerator / _denominator`, is intuitive for the students. However it is shown using a test program that the fractions ½ and ¼ both evaluate to 0.0. This is due to the fact that integers are closed under division in Java. Therefore the solution using a type-cast is presented, illustrating mixed-type calculations.

3.2.10 Counter

3.2.10.1 Concepts Covered

Round-off error.

3.2.10.2 Description

A Java application that displays a running count is shown. The code for the relevant part of the program is shown in Figure 3.11. A `CountBehavior` object is created with an initial value of 0.0 and an increment of 0.1.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
package Counter;

/**
 * CountBehavior.java
 * @author Philip R Ventura
 */

public class CountBehavior extends NGP.Timer{

   /**
    * The label to display on.
    */
   private NGP.Components.Label _label;

   /**
    * The current value.
    */
   private double _value;

   /**
    * The increment to _value.
    */
   private double _increment;

   public CountBehavior (NGP.Components.Label label, double
                           value, double increment){
      super(500);  // call activate() every 500
                   // milliseconds
      _label = label;
      _value = value;
      _increment = increment;
   }

   /**
    * Called every 500 milliseconds to change the display
    */
   public void activate() {
      _value = _value + _increment;
      _label.setText("" + _value);
   }
}// CountBehavior
```

Figure 3.11 Code for Demonstrating Round-Off Error

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

3.2.10.3 Discussion

Upon running the program, the first two numbers appear as expected, namely 0.1 and 0.2. At the next step, 0.30000000000000004 is displayed, then 0.4. The next deviation occurs after 0.7, namely 0.7999999999999999 is displayed instead of 0.8. Follow-up calculations are therefore off.

Students get a first-hand look at the problem of round-off error. To shed light on the topic, the conversion of 0.1 to binary is done in class, and students observe that the process leads to non-terminating sequence. Since there are only a finite number of bits for the machine's representation of a double, the binary representation of 0.1 is only an approximation.

3.2.11 Grade

3.2.11.1 Concepts Covered

`if` and `switch...case` statements, invariants.

3.2.11.2 Description

Students are asked to consider the design of a grade class that allows for creation using an integer raw score. Then they are asked to consider writing the code to convert between the integer score and a letter grade (without pluses or minuses). Finally, students are asked to write the code to convert the raw score to a number of quality points.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

3.2.11.3 Discussion

The conversion of the raw score to a letter grade is a straight forward application of cascading if-else statements. While the code for the conversion from raw score to quality points could be coded in the same manner, students are shown the code to convert between a letter grade and the quality points using a `switch...case` statement. The limitations of `switch...case` in Java are discussed. The difference in terms of number of comparisons needed is also highlighted.

3.2.12 Password

3.2.12.1 Concepts Covered

Object equality vs. object identity.

3.2.12.2 Description

Students are shown a graphical application that prompts the user for a password. The user is required to type the password into a text box. The students are then asked to write the code to validate the password.

3.2.12.3 Discussion

The students are shown the method in which they must place the password validation code, namely `returnPressed` (see Figure 3.12). They quickly realize the need for an `if` statement. The first attempt is to have the condition for the if statement read as

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

`userInput == _password`. The program is then run and tried. Much to the students' surprise, the message "Access granted" is never displayed even upon entering the proper password.

```
public void returnPressed() {
   // validate password here
  String userInput = this.getText();
  if ( ___________________________ ) {
    _passPanel.showMessage("Access granted.");
  } // end of if ()
  else {
    _passPanel.showMessage("Access denied.");
  } // end of else
```

Figure 3.12 Password Validation Code

The students are then told of the fact that the == operator in Java tests to see if two references refer to the same object (object identity) rather than testing to see if two objects are equal. At that point the `equals` method is introduced.

3.2.13 Turtle

3.2.13.1 Concepts Covered

Definite vs. indefinite iteration, and entry- vs. exit-test loops.

3.2.13.2 Description

Students are shown a graphical application that displays controls for a Logo-style turtle. There are three buttons that do not work: Random Walk, Draw Square, and Filled Square. The students need to write the code for each of the buttons. The random walk

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

should have a turtle pick a random pen color, turn a random amount, and move a random number of steps forward.

3.2.13.3 Discussion

For drawing a square, students are asked to first consider the series of steps, namely telling the turtle to move forward and then turn 90 degrees and do this four times. The code is written by cutting and pasting the move and turn commands. Then students are introduced to the concept of definite iteration and the Java `for` loop. The square code is rewritten to use a `for` loop. The code for drawing a filled square also makes use of a `for` loop.

Then attention is turned to random walk. Students are warned of the possibility that the turtle could simply wander off the screen. They are then asked how many iterations it will take before the turtle moves off the screen. They quickly realize that there is no answer. The Java `while` and `do...while` loops are then introduced along with the difference between entry- and exit-test loops. Students are then asked which loop is a better choice for random walk or if they are equivalent in that context. Some students vote for equivalence, and the possibility of the turtle already being off the screen when the Random Walk button is clicked is brought to their attention.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

3.2.14 Bouncing

3.2.14.1 Concepts Covered

Use of generics, `ArrayList` as a bag.

3.2.14.2 Description

Students are asked to design a program with multiple bouncing balls, which not only bounce of the sides, top and bottom of the screen, but also bounce off of one another.

3.2.14.3 Discussion

See Figure 3.13 for the `BouncingBall` implementation. At this point in the course, the students are quite comfortable with extending classes and overriding methods to enhance functionality. So the idea of creating a new ball class is trivial. When asked, the students generally realize that each ball must have knowledge of the others.

Furthermore, the notion of using a class-level (static) collection of the balls is easily appreciated. Otherwise, every time a new ball is created all of the current balls would need to be notified. Since objects share a class level variable, all balls will know of each other.

The first attempt at the code for `update` omits the selection statement in the loop. When the program is run the balls “shudder” in place. Then the discussion reminds students that the list of balls contains all the balls, so therefore each ball intersects itself.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
package Bouncing; // Generated package name

/**
 * BouncingBall.java
 *
 *
 * Created: Wed Apr 16 16:15:35 2003
 *
 * @author Philip R Ventura
 */

public class BouncingBall extends CSE115.BallWorld.Ball{

  private static final java.util.ArrayList<BouncingBall>
    BALL_LIST = new java.util.ArrayList<BouncingBall>();

  public BouncingBall (){
    BALL_LIST.add(this);
  }

  public void update() {
    super.update();

    java.util.Iterator<BouncingBall> iter =
      BALL_LIST.iterator();
    BouncingBall ball;

    while ( iter.hasNext() ) {
      ball = iter.next();
      if ( ball != this && this.intersects(ball) ) {
        this.setVelocity(-this.getDx(), -this.getDy());
        super.update();       // update position of ball
        break;
      } // end of if ()
    } // end of while ()
  }

}// BouncingBall
```

Figure 3.13 Code for Bouncing Balls

The code introduces a new feature of Java to be added in Java version 1.5, namely generics a.k.a. parametric polymorphism. In this course, student exposure to generics is

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

limited to the use of the collection classes. In general, students show no problem with the concept.

3.2.15 TicTacToe

3.2.15.1 Concepts Covered

`HashMap`: the ordinary person's array.

3.2.15.2 Description

The students are given a partially completed framework for a graphical two-player Tic-Tac-Toe game. Students are asked to implement the game logic, that is, allow for marks to be placed, and check for draw and win conditions.

3.2.15.3 Discussion

The focus of this example is the use of `HashMap` to model a grid, namely the Tic-Tac-Toe board. The Tic-Tac-Toe board is essentially a `HashMap` indexed by row/column pairs which stores marks, X's or O's. This design uses the `HashMap` as a replacement for 2-D arrays. The workhorse of the design is the `Position` class that models the row/column pair. The code for the `Position` class is given in Figure 3.14. The `Position` class takes care of translations between row/column and screen coordinates (see the second constructor and `toPoint` method).

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
package TicTacToe; // Generated package name

/**
 * Position.java
 *
 *
 * Created: Fri Nov 22 12:23:16 2002
 *
 * @author Philip R Ventura
 */

public class Position implements BoardConstants {
  private int _row;
  private int _col;

  public Position (int row, int col){
    _row = row;
    _col = col;
  }

  public Position(java.awt.Point pt) {
    _row = pt.y / SQUARE_SIZE;
    _col = pt.x / SQUARE_SIZE;
  }

  public int getRow() {
    return _row;
  }

  public int getCol() {
    return _col;
  }

  public String toString() {
    return "[" + _row + "," + _col + "]";
  }

  public java.awt.Point toPoint() {
    return new java.awt.Point(_col * SQUARE_SIZE,
                              _row * SQUARE_SIZE);
  }

  public int hashCode() {
```

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
    return this.toPoint().hashCode();
  }

  public boolean equals(Object obj) {
    if ( obj == null || this.getClass() != obj.getClass() )
{
      return false;
    } // end of if ()
    Position other = (Position) obj;
    return _row == other._row && _col == other._col;
  }
}// Position
```

Figure 3.14 Position Class Code

A crucial part of the implementation of the `Position` class are the `equals` and `hashCode` methods. Both are essential for allowing `Position` objects to be used as keys for the `HashMap` class. Figure 3.15 shows the code for `TicTacToeBoard` that uses the `Position` class along with the `HashMap`. To conserve space the code for drawing lines and the code for checking columns and diagonals has been omitted. The code is provided to show how a `HashMap` can be used in lieu of a 2-D array.

```
package TicTacToe; // Generated package name

/**
 * TicTacToeBoard.java
 *
 *
 * Created: Fri Nov 22 12:06:08 2002
 *
 * @author Philip R Ventura
*/

public class TicTacToeBoard
         extends NGP.Containers.ReactiveDrawingPanel
         implements BoardConstants {

  private java.util.HashMap<Position, Mark> _board;
```

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
  private TicTacToeGame _game;
  private BoardState _state;

  public TicTacToeBoard (TicTacToeGame game){
    super(game);
    _game = game;
    this.setDimension(new java.awt.Dimension(3 *
                                            SQUARE_SIZE,
                                            3 *
                                            SQUARE_SIZE));
    this.setColor(java.awt.Color.white);
    this.reset();
  }

  public void reset() {
    this.removeAllGraphics();
    _board = new java.util.HashMap<Position, Mark>();
    this.drawLines();
    _state = new PlayBoardState();
  }

  public void react(java.awt.Point pt) {
    _state.boardClicked(this, pt);
  }

  public void placeMark(Position pos) {
    if ( _board.containsKey(pos) ) {
      _game.showMessage("Already a mark there!");
      return;
    } // end of if ()

    Mark m = new Mark(this, _game.getPlayer());
    m.setPosition(pos);
    _board.put(pos, m);
    if ( this.isFull() || this.playerWon(_game.getPlayer())
) {
      _state = new NullBoardState();
    } // end of if ()

    _game.playerMoved();
  }

  public boolean playerWon(char playersMark) {
```

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

```
    for ( int row = 0; row < 3; row++ ) {
      if ( this.rowWin(row, playersMark) ) {
          return true;
      } // end of if ()
    } // end of for ()
  }

  private boolean rowWin(int row, char playersMark) {
    Mark m;
    for ( int col = 0; col < 3; col++ ) {
      if ( !_board.containsKey(new Position(row, col)) ) {
     // no mark at that location
          return false;
      } // end of if ()
      m = _board.get(new Position(row, col));
      if ( playersMark != m.getMark() ) {
          return false;
      } // end of if ()
    } // end of for ()
    return true;
  }

  public boolean isFull() {
    return _board.size() == 9;
  }
}// TicTacToeBoard
```

Figure 3.15 TicTacToeBoard HashMap-based Implementation

3.3 Conclusion

This chapter has provided a model curriculum for the graphical design-centric objects-first CS1 along with exemplars used for its teaching. The next three chapters attempt to answer the question of how predictors of success for this course are different from those of traditional CS1's.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Chapter 4

Experimental Method

This chapter discusses the methodology utilized in identifying the predictors of success for the graphical objects-first design-centric CS1 discussed earlier. Section 4.1 outlines the research questions to be investigated by the experimental work of the next chapter. Section 4.2 provides details on the subjects involved in the study. Section 4.3 describes the testing instruments used to gather data, while Section 4.4 presents the manner in which data were collected and calculated. Finally, Section 4.5 identifies and defines both the independent and the dependent variables used for analysis.

4.1 Research Questions

The experiments were run to answer a number of exploratory questions. The questions and variables were motivated by the prior research on traditional CS1 courses.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

4.1.1 Traditional Predictors

1) What is the relationship of mathematical ability to OF CS1?

2) What is the effect of prior programming experience for success in an OF CS1?

4.1.2 Success

3) What is the effect of gender on success?

4) What is the effect of year in school on success?

5) What is the effect of major/intended major on success?

6) What is the effect of reason for CS major on success?

7) What variables are most important among credit hours, hours worked at job, high school average, high school rank, number of years of high school math, SAT math, SAT verbal, number of office hour visits, recitation attendance, comfort level, critical thinking ability, programming self-efficacy, attribution for success/failure, prior programming experience, number of labs submitted, and number of exams taken for success?

4.1.3 Resign rates

8) Is there any gender bias evident in terms of resign rates?

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

9) Are there any differences between students who resign and those who do not in terms of number of credit hours registered for?

10) What is the effect of year in school on resign rates?

11) What is the effect of major/intended major on resign rates?

4.2 Subjects

Subjects were students enrolled in CSE115 at University at Buffalo, SUNY, from the Spring 2002, Fall 2002, and Spring 2003 semesters. Institutional Review Board (IRB) approval was obtained prior to data collection for the study. Participation was entirely voluntary. During the first semester, students were not offered extra credit for participation. Due to low response rate that semester, extra credit was offered in the subsequent two semesters for study participation. Overall 50% of students completed the questionnaire. As per federal guidelines, alternate extra credit opportunities were available for those who were unwilling or unable to participate.

Only the data for those students who were taking CSE115 for the first time was used in analysis. This was to guard against practice effect.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

4.3 Materials

4.3.1 Cornell Critical Thinking Test

The Cornell Critical Thinking Test (Level Z) developed by Robert Ennis (1985) was used to assess students' critical thinking ability. Ennis defines critical thinking as "the process of reasonably deciding what to believe and do," (Ennis, 1985). It incorporates aspects such as induction, deduction, observation, credibility, and assumptions.

4.3.2 Computer Programming Self-Efficacy Scale

The Computer Programming Self-Efficacy Scale as developed by Ramalingam & Wiedenbeck (1998) was used to measure self-efficacy, i.e., students' beliefs about their own programming ability. It was chosen since past work (Cantwell Wilson & Shrock, 2001) also included it, and it therefore serves as a convenient comparison. The test measures four different dimensions of self-efficacy (all quotes in the list below were taken from (Ramalingam & Wiedenbeck, 1998)):

1. *Simple programming* – concerns "self-efficacy for carrying out elementary-to-intermediate level activities (writing statements, blocks of code, small programs, and moderate-sized programs)." (p. 373)

2. *Complex programming* – concerns "designing, comprehending, modifying, and debugging long, complex programs, reusing code written by others." (p. 373)

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

3. *Self-regulation* – concerns "ability to control and regulate self to achieve an end (e.g., ability to complete a programming project under time pressure or if one has no interest in it.)" (p. 373)

4. *Independence/persistence* – "concerns ability to act independently and to persevere in the face of different conditions varying in adversity." (p. 373)

4.4 Procedure

4.4.1 Paper and Electronic Testing

The first semester of data collection used paper-based data collection only. However, in an effort to encourage more students to participate through ease of administration, the questionnaire was converted to electronic form. A web application was written, by Christopher A. Egert, Ph.D., using primarily ASP and XML to administer the questionnaire. The Cornell Critical Thinking Test (Level Z) was added at this stage. The ASP/XML based online survey was setup to allow for the Cornell Critical Thinking Test to be timed as per the testing instructions.

Data collection this second semester, proceeded just after the students received their first exam grades, since the questions on attribution deal with the first exam.

In an effort to capture data on students who resign, the ASP/XML online survey was reconfigured to allow data collection to occur in two phases. The first phase gathered basic demographics as well as Cornell Critical Thinking score, while the second phase

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

gathered self-efficacy and attribution of success/failure scores. The decision to gather self-efficacy in the second stage was to help guard against some students' mistaken belief that they understood OOP/OOD at the beginning of the course. Unfortunately, due to problems in the development stage of this new version of the ASP/XML survey, data collection did not begin until the fifth week of classes (the week before the first exam). Furthermore, since data collection continued until the end of the semester, this seemed to cause many students to delay taking the survey. Despite the additional effort to gather data on students who might resign, only five students who resigned actually took the survey. Therefore analysis of the differences between students who resigned and those who didn't was limited to the data that could be obtained from InfoSource (see Section 4.4.3).

Despite the slight differences in data collection methods among the three semesters, it is not believed that these differences fundamentally changed the data that were collected.

4.4.2 Grades

Student grades were collected from the individual grade books for each course. The course grade was computed as the weighted average of three components: homeworks & quizzes (10%), lab average (40%), and exam component (50%). In the first two semesters, there were eight lab assignments; the last semester only had seven. There are two ways in which the exam component is computed. If a student has taken all three exams, then *exam component* = max(((.15 * *exam 1 score* + .15 * *exam 2 score* + .20 *

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

final exam score) / 50) * 100, *final exam score*). Otherwise, *exam component* = ((.15 * *exam 1 score* + .15 * *exam 2 score* + .20 * *final exam score*) / 50) * 100.

While individual course instructors wrote the exams, the exams were largely equivalent in terms of the material tested.

4.4.3 InfoSource Data Collection

The University at Buffalo, SUNY, maintains a database, InfoSource, containing a wealth of information on students and the University proper, including student and applicant demographics and application materials, enrollment statistics (past and present), etc.

Microsoft® Access was used with Oracle's (8i) ODBC drivers to obtain information on a number of demographic variables (see Section 4.5) from InfoSource. Unfortunately, information on high school average and high school percentile rank was not available for all students. A similar but less severe problem was also noted with SAT scores.

4.4.4 Last Log Mining

In an attempt to accurately measure student recitation attendance, the Unix `last` command was used. The `last` command provides information for when a person logged on and logged off a particular machine. For the Spring 2002 semester, the `last` command was run for each Sun Microsystems machine in the lab to capture full information on logins. This information was imported into Microsoft® Access 2000

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

(9.0.3821 SR-1) where further manipulation for the extraction of student logins during recitations occurred. In the Fall 2002 semester, the `last` command only needed to be run on the undergraduate server since the class now used a different lab that only made use of Sun Microsystems terminals.

Data collection in the Spring 2003 semester was complicated by the lab terminals having been upgraded to Sun Microsystems Sun Ray machines. The usual run of the `last` command produced many zero-minute-long sessions. The Sun Rays made it seem that all logins occurred from the same terminal, `dtlocal`. The `last` command was being "lazy" in its reporting by showing for login durations the difference between one login on `dtlocal` and the previous login on `dtlocal`. It was discovered that this was not an issue if `last` was run given an individual student's name. Therefore a shell script was created to read a file containing the usernames of all students in the class and run last for each concatenating the results to one file.

4.4.5 Office Hour Tracker

A Java application was created to track student use of office hours. All CSE115 staff were required to use the program during office hours. It stored in an Oracle database using JDBC the student's username, the faculty member's username, the date and time of the visit, the duration of the visit and the reason for the visit (Lab, Coursework, Homework, Grading, and Other as a free text field). A bug appeared in the latest version

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

that caused the duration of many visits to appear as zero. Therefore, the number of visits was used in the analysis rather than the amount of time spent.

4.4.6 Analyses

All analyses were carried out using SPSS 11.5 for Windows. As per the American Psychological Association Publication Manual (American Psychological Association., 2001), all tests were run with $\alpha = .05$, unless noted otherwise.

4.5 Variables

This section details the independent and dependent variables used in the experimental work as well as their source.

4.5.1 Independent Variables

1) *Gender* – dichotomous variable denoting the student's sex; obtained from InfoSource
2) *Year in school* – nominal variable denoting student's year in school, with value NM for students who are non-matriculated; obtained from InfoSource
3) *Major/intended major* – nominal variable denoting the major, or intended major where no major exists for the student; obtained from InfoSource

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

4) *Credit hours* – continuous variable denoting the number of credit hours a student was registered for; obtained from InfoSource

5) *Num hours per week at job* – continuous variable denoting the number of hours per week on average that a student worked at a job; obtained from questionnaire

6) *HS avg* – continuous variable denoting the student's high school average; obtained from InfoSource

7) *HS percentile rank* – continuous variable denoting the student's percentile in high school; obtained from InfoSource

8) *Years HS math* – continuous variable denoting the number of years of high school math a student took; obtained from questionnaire

9) *SAT math* – continuous variable, a record of the student's SAT math score; obtained from InfoSource

10) *SAT verbal* - continuous variable, a record of the student's SAT verbal score; obtained from InfoSource

11) *Office hour visits* – continuous variable, the number of times a student came to office hours during the semester for non-grading reasons; obtained from the *Office Hours Log* database

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

12) *Percent recitation time usage* – continuous variable, the percentage of recitation time a student was in recitation (assuming 1 recitation per week); calculated from Unix `last` log data

13) *Reason for CS major* – nominal variable denoting the reason a student was a computer science major/intended major; obtained from questionnaire

14) *Comfort level* – continuous variable denoting a student's comfort level in class; based on the questionnaire by Cantwell Wilson & Shrock (2001), it includes measures of programming comfort level as well as comfort for asking questions and going to office hours; obtained from questionnaire

15) *Cornell CT score* – continuous variable denoting a student's critical thinking ability, measured by the Cornell Critical Thinking Test (Level Z); obtained from questionnaire

16) *SE-independence/persistence* – continuous variable denoting self-efficacy along the independence/persistence dimension as measured by the Computer Programming Self-Efficacy Scale; obtained from questionnaire

17) *SE-complex programming* - continuous variable denoting self-efficacy along the complex programming dimension as measured by the Computer Programming Self-Efficacy Scale; obtained from questionnaire

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

18) *SE-self-regulation* - continuous variable denoting self-efficacy along the self-regulation dimension as measured by the Computer Programming Self-Efficacy Scale; obtained from questionnaire

19) *SE-simple programming* - continuous variable denoting self-efficacy along the simple programming dimension as measured by the Computer Programming Self-Efficacy Scale; obtained from questionnaire

20) *Att-exam difficulty* – ordinal variable measuring attribution of success/failure (on first exam) to difficulty; obtained from questionnaire

21) *Att–luck* - ordinal variable measuring attribution of success/failure (on first exam) to luck; obtained from questionnaire

22) *Att–effort* - ordinal variable measuring attribution of success/failure (on first exam) to effort; obtained from questionnaire

23) *Att–ability* - ordinal variable measuring attribution of success/failure (on first exam) to ability; obtained from questionnaire

24) *C (in years)* – continuous variable denoting the number of years a student had been programming in C prior to taking CSE115; obtained from questionnaire

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

25) *C++ (in years)* - continuous variable denoting the number of years a student had been programming in C++ prior to taking CSE115; obtained from questionnaire

26) *Java (in years)* - continuous variable denoting the number of years a student had been programming in Java prior to taking CSE115; obtained from questionnaire

27) *Pascal (in years)* - continuous variable denoting the number of years a student had been programming in Pascal prior to taking CSE115; obtained from questionnaire

28) *Basic (in years)* - continuous variable denoting the number of years a student had been programming in Basic prior to taking CSE115; obtained from questionnaire

29) *HTML (in years)* - continuous variable denoting the number of years a student had been programming in HTML prior to taking CSE115, used to filter out students who report prior programming experience while only having experience with HTML; obtained from questionnaire

30) *Scripting (in years)* - continuous variable denoting the number of years a student had been programming in scripting languages prior to taking CSE115; obtained from questionnaire

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

31) *Opt lang 1 – 3* - nominal variables denoting the languages, that were not specified explicitly in the questionnaire, a student had been programming in prior to taking CSE115; obtained from questionnaire

32) *Opt lang 1 – 3 (in years)* - continuous variables denoting the number of years a student had been programming in languages, that were not specified explicitly in the questionnaire, prior to taking CSE115; obtained from questionnaire

33) *Ttl yrs prog* – continuous variable denoting the total number of years a student had been programming prior to CSE115 (does not include HTML); calculated from data obtained on questionnaire

34) *Prior programming experience* – a dichotomous variable indicating whether a student had programming experience (not including HTML) prior to CSE115, created to correct for any self-report measurement errors for prior programming experience; computed from data obtained on questionnaire

35) *Num prog lang* – a continuous variable denoting the total number of programming languages (sans HTML) that a student had experience with prior to CSE115; computed from data obtained on questionnaire

36) *Lab 2 – 8 time (in hours)* – continuous variables denoting the amount of time a student spent working on each of these labs; ultimately dropped

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

from the analysis due to the incomplete nature of this data; obtained from student self-report data

37) *Percent labs submitted* – continuous variable denoting the percent of labs that a student submitted in the course; calculated from the grade book data

38) *Num exams taken* – continuous variable denoting the number of exams a student took in the course; calculated from the grade book data

39) *Resigned* – dichotomous variable indicating whether a student resigned the course; obtained from InfoSource

4.5.2 Dependent Variables

40) *Lab avg* – continuous variable denoting the student's lab average for CSE115; obtained from the grade book

41) *Exam avg* - continuous variable denoting the student's exam average for CSE115; obtained from the grade book

42) *Course avg* - continuous variable denoting the student's course average; obtained from the grade book

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Chapter 5

Experimental Results

This chapter reports on the results of the statistical analyses conducted to test each of the research questions listed in Section 4.1. Section 5.1 reports on the analyses undertaken to examine questions dealing with the three measures of success, namely, course average, exam average, and lab average. Section 5.1.1 examines the question of whether prior programming experience is a predictor of success. The effect of gender on success is tested in Section 5.1.2. Section 5.1.3 describes the results of tests for differences in performance based on a student's year in school. Sections 5.1.4 and 5.1.5 test for differences in success by major/intended major and reason for selecting computer science as a major, respectively. To answer the question of, "what best predicts success?" Section 5.1 concludes with the results of multivariate analysis into predictors of success for the objects-first CS1 (§5.1.6).

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Results of investigations into rate of resignation is provided in Section 5.2. Gender (§5.2.1), credit hours students were registered for at the beginning of the semester (§5.2.2), year in school (§5.2.3), and major/intended major (§5.2.4) are all considered.

Statistical inference, testing assumptions, parametric vs. non-parametric testing, and multivariate analysis are not explained in this dissertation. The author has found the following statistics texts to be indispensable (Aron & Aron, 2002; Norusis, 2002).

5.1 Success

This section reports results of tests of the three measures of success namely, course average, lab average, and exam average.

5.1.1 Prior Programming Experience

In an attempt to answer the question of the effect of prior programming experience on success, a comparison of those students with prior programming experience versus those students without prior programming experience in terms of course success was conducted. These analyses considered the following types of prior programming experience:

1) Prior experience programming in any language (except HTML)

2) Prior experience with object-oriented languages, namely, C++ and Java

3) Self-efficacy for object-orientation

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

5.1.1.1 Programmers vs. Non-programmers

For this analysis students who participated in the surveys were divided into two groups, one with and one without prior programming experience. This technique has the advantage of compensating for measurement errors for the self-reported prior programming experience data.

An a priori analysis of the data revealed that the measures of course success were not distributed normally for the dichotomous prior programming experience variable. Therefore a Mann-Whitney U was employed to test for differences between the two groups (see Table 5.1).

	Lab Avg	Exam Avg	Course Avg
Mann-Whitney U	4287.500	4163.000	4302.000
Wilcoxon W	5368.500	5244.000	5383.000
z	-1.193	-1.466	-1.161
p	.23298	.14253	.24568

Table 5.1 Mann-Whitney U for Success by Prior Programming Experience

The Mann-Whitney U tests failed to uncover differences on the measures of success between the programmers and non-programmers. This was also the case with the parametric tests.

5.1.1.2 Experience with OO Languages

For this analysis, C++ and Java experience were treated as dichotomous variables. The test assumptions for independent samples *t*-test were met. Table 5.2 shows the

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

results of the independent samples t-tests for success by prior C++ experience. As can be seen from the table, there were no significant differences for any of the measures of success.

	t-test for Equality of Means						
	t	*df*	*p*	Mean Difference	Std. Error Difference	95% Confidence Interval of the Difference	
						Lower	Upper
Lab Avg	-.893	254	.37263	-2.68	3.003	-8.597	3.232
Exam Avg	-1.187	254	.23621	-3.062	2.5791	-8.1414	2.0170
Course Avg	-1.142	254	.25457	-2.81	2.460	-7.655	2.036

Table 5.2 t-tests for Success by Prior C++ Experience

A similar analysis was undertaken for prior Java experience. Table 5.3 shows the results of theses tests. Upon first inspection, there appear to be no significant differences. However, the tests were conducted using a 2-tailed analysis. A 1-tailed analysis would divide the *p* values by 2. When this is done, the mean difference for exam average is significant ($p = .03898$). In other words, there are differences in the exam averages between those students with prior programming experience in Java and those without it. An examination of means for each group revealed surprisingly, that students without prior Java programming experience did better than those who had programmed in Java prior to CS1!

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

	t-test for Equality of Means						
	t	*df*	*p*	Mean Difference	Std. Error Difference	95% Confidence Interval of the Difference	
						Lower	Upper
Lab Avg	.039	254	.96867	.14	3.584	-6.917	7.199
Exam Avg	1.770	254	.07795	5.421	3.0627	-.6109	11.4520
Course Avg	1.626	254	.10516	4.75	2.924	-1.004	10.513

Table 5.3 t-tests for Success by Prior Java Experience

5.1.1.3 Self-Efficacy for Object-Orientated Concepts

To test the students' prior exposure to object-orientation the Computer Programming Self-Efficacy Scale (Ramalingam & Wiedenbeck, 1998) was used. It measures a student's beliefs about his or her own programming ability. It includes three questions that deal object-orientation, namely:

1) "I understand the object-oriented paradigm." (Ramalingam & Wiedenbeck, 1998)

2) "I can identify the objects in the problem domain and declare, define, and use them." (Ramalingam & Wiedenbeck, 1998)

3) "I can make use of a class that is already defined, given a clearly labeled declaration of the class." (Ramalingam & Wiedenbeck, 1998)

Student responses on the three OO self-efficacy questions were averaged, and represented by the variable named SE-OO. The Pearson correlation coefficient was computed for SE-OO with each of the measures of course success (see Table 5.4).

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

	SE - OO Programming	Lab Avg	Exam Avg	Course Avg
r	1	-.053	.055	.036
p (1-tailed)	.	.20280	.19315	.28450

Table 5.4 Pearson Correlations for OOP Self-Efficacy and Success

These tests fail to show a significant effect. That is, students' beliefs about their ability to program object-orientedly are not correlated with any measure of course success.

5.1.2 By Gender

An investigation of gender bias in the OF CS1 was conducted by performing *t*-tests for the three measures of success. Test assumptions of normality and equality of variance were verified. While normality was violated by the male group, it was not deemed severe enough to affect the validity of the *t*-test. The results are presented in Table 5.5. The tests fail to reveal any gender bias for course success. Given the violation of the normality assumption, the non-parametric Mann-Whitney U tests were also run and likewise failed to show a difference for any of the measures of success.

	t-test for Equality of Means						
	t	*df*	*p*	Mean Difference	Std. Error Difference	95% Confidence Interval of the Difference	
						Lower	Upper
Lab Avg	-1.271	378	.20458	-5.58	4.393	-14.222	3.055
Exam Avg	-1.339	378	.18141	-5.248	3.9195	-12.9545	2.4589
Course Avg	-1.686	378	.09272	-6.55	3.883	-14.180	1.090

Table 5.5 t-test for Success by Gender

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

5.1.3 By Year in School

In order to investigate the effect of year in school on success, a one-way ANOVA was planned for course average, lab average, and exam average with the independent variable being year in school. Since there were two masters-level students and one doctorate-level student, these groups were combined to a single graduate group. An a-priori verification of ANOVA test assumptions revealed that each of the measures of success were not distributed normally across year in school. Therefore, the ANOVA was abandoned in favor of the non-parametric Kruskal-Wallis test. Table 5.6 shows the mean ranks for the different measures of success by year in school.

	Year in School	n	Mean Rank
Lab Avg	FR	196	188.66
	SO	71	169.53
	JR	60	186.95
	SR	40	235.98
	GR	3	279.33
	NM	10	188.30
	Total	380	
Exam Avg	FR	196	184.40
	SO	71	177.00
	JR	60	191.54
	SR	40	239.35
	GR	3	254.17
	NM	10	185.25
	Total	380	
Course Avg	FR	196	189.33
	SO	71	171.08
	JR	60	183.75
	SR	40	233.53

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

GR	3	276.83
NM	10	193.85
Total	380	

Table 5.6 Mean Ranks for Success by Year in School

The Kruskal-Wallis analysis of variances revealed statistical significance only with regard to the lab average dependent variable (see Table 5.7).

	Lab Avg	Exam Avg	Course Avg
χ^2	11.529	10.626	10.469
df	5	5	5
p	.04184	.05931	.06300

Table 5.7 Kruskal-Wallis for Success by Year in School

Post-hoc application of pair-wise Mann-Whitney *U* tests with Bonferroni error-correction (α = .05 / 15 comparisons = .0033) revealed that the only statistically significant difference in mean ranks occurs between sophomores and seniors (see Table 5.8), that is sophomore students' lab grades were lower than those of seniors.

	Lab Avg
Mann-Whitney U	916.500
Wilcoxon W	3472.500
z	-3.093
p	.00198

Table 5.8 Mann-Whitney U for Lab Average between Seniors and Sophomores

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

5.1.4 By Major

An examination of the effect of major/intended major on the three measures of success is reported in this section. Table 5.9 shows the abbreviations for each of the majors reported in the course.

Abbreviation	Major
ARC	Architecture
ASE	Aerospace Engineering
BCH	Medical and Biomedical Sciences
BIO	Biology
CE	Chemical Engineering
CEN	Computer Engineering
CHE	Chemistry
CIE	Civil Engineering
CPP	Computational Physics
CS	Computer Science
DAC	Dance
DMS	Media Studies
EAS	Engineering and Applied Science
ECO	Economics
EE	Electrical Engineering
ENG	English
FA	Fine Arts
GEO	Geography
HIS	History
ME	Mechanical Engineering
MG	Management
MGA	Management – Accounting
MTH	Math
PHY	Physics
PSY	Psychology
SSC	Interdisciplinary Social Sciences
TH	Theater
UNM	Unmatriculated

Table 5.9 Major Abbreviations

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Given that a number of majors had only a single student, the majors/intended major of the students were reclassified based on type. The types are ART (Art), BUS (Business), CEN (Computer Engineering), CS (Computer Science), ENG (Engineering), HUM (Humanities), MTH (Math), SCI (Science), and UNM (Unmatriculated). The CEN and CS majors remain unchanged as they are the intended audience of the course. The MTH major remains unchanged as it relates back to the original question of the effect of mathematical ability on an OF CS1. The transformation is listed in Table 5.10.

Major (by type)	Majors
ART	DAC DMS FA TH
BUS	ECO MG MGA
CEN	CEN
CS	CS
ENG	ARC ASE CE CIE EAS EE
HUM	ENG HIS
MTH	MTH
SCI	BCH BIO CHE CPP

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

	GEO PHY PSY SSC
UNM	UNM

Table 5.10 Major to Major (by type) mapping

While this transformation helped the problem of singleton groups as well as lessens information overload by compacting the 28 original categories to only nine, the scores for all three measures of success were non-normally distributed across all majors and did not reflect the same degree of skew. Therefore a Kruskal-Wallis ANOVA (rather than the parametric, one-way ANOVA) was selected to test for differences in success by major. The mean ranks for success by type of major/intended major are shown in Table 5.11.

	Major (by Type)	*n*	Mean Rank
Lab Avg	ART	13	195.92
	BUS	11	236.09
	CEN	86	175.13
	CS	193	192.41
	ENG	23	147.72
	HUM	4	75.63
	MTH	3	213.67
	SCI	20	200.45
	UNM	19	194.08
	Total	372	
Exam Avg	ART	13	202.42
	BUS	11	216.45
	CEN	86	182.21
	CS	193	190.46
	ENG	23	160.63
	HUM	4	56.00
	MTH	3	209.33

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

	SCI	20	195.73
	UNM	19	182.95
	Total	372	
Course Avg	ART	13	186.62
	BUS	11	225.95
	CEN	86	179.63
	CS	193	192.00
	ENG	23	154.61
	HUM	4	58.13
	MTH	3	212.33
	SCI	20	190.40
	UNM	19	196.24
	Total	372	

Table 5.11 Mean Ranks for Success by Type of Major/Intended Major

Table 5.12 shows the results of the Kruskal-Wallis ANOVAs. The tests fail to support the idea that success has any relation to type of major/intended major. These results were consistent with the findings of the parametric test.

	Lab Avg	Exam Avg	Course Avg
χ^2	11.850	9.064	10.418
df	8	8	8
p	.15800	.33696	.23690

Table 5.12 Kruskal-Wallis for Success by Type of Major/Intended Major

5.1.5 By Reason for CS Major

At the outset of the research it was hypothesized that the reason a student had for majoring in computer science would have an impact on that student's success in the course. Given the small numbers of students in a number of the categories as well as a non-normal distribution of the measures of success by the reason for CS major/intended

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

major, the use of a one-way ANOVA was prohibited. Instead the Kruskal-Wallis ANOVA was selected as the more appropriate test. Table 5.13 shows the mean ranks for each of the reasons for CS major/intended major.

	Reason for CS Major	*n*	Mean Rank
Lab Avg	friendinfield	15	127.50
	games	43	105.58
	hsguidance	3	163.33
	interest	66	101.60
	parent	2	160.00
	payscale	21	96.38
	programming	25	126.54
	unsure	35	85.70
	Total	210	
Exam Avg	friendinfield	15	122.50
	games	43	88.60
	hsguidance	3	136.83
	interest	66	114.58
	parent	2	169.00
	payscale	21	87.31
	programming	25	125.20
	unsure	35	92.39
	Total	210	
Course Avg	friendinfield	15	129.73
	games	43	98.41
	hsguidance	3	135.83
	interest	66	108.86
	parent	2	172.75
	payscale	21	87.21
	programming	25	124.82
	unsure	35	88.21
	Total	210	

Table 5.13 Mean Ranks for Success by Reason for Computer Science Major/Intended Major

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

The Kruskal-Wallis ANOVA results are reported in Table 5.14. The tests fail to support the hypothesis that reason for computer science major/intended major has an impact on success.

	Lab Avg	Exam Avg	Course Avg
χ^2	13.751	15.092	13.634
df	7	7	7
p	.05580	.03484	.05809

Table 5.14 Kruskal-Wallis for Success by Reason for Computer Science Major/Intended Major

5.1.6 Stepwise Multiple Linear Regression

The analyses in this section were run to examine what variables are most important among credit hours, hours worked at job, high school average, high school rank, number of years of high school math, SAT math, SAT verbal, number of office hour visits, recitation attendance, comfort level, critical thinking ability, programming self-efficacy, attribution for success/failure, prior programming experience, number of labs submitted, and number of exams taken for success.

5.1.6.1 Course Average

Prior to the multiple regression, scatterplots and Pearson correlation coefficients were computed for each of the independent variables, with course average to guard the assumption of linear regression that a linear relationship does in fact exist between the each independent variable and the dependent variable (see Appendix C).

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

On examining the scatterplots, the need for logarithmic transformation of number of office hour visits and percent recitation utilization became apparent. The function $ln(1 + x)$ was applied to each variable for the current analysis. Only those variables which revealed statistically significant values for Pearson correlation coefficients were used in building the model for course average. These variables are: number of credit hours taken during the current semester, high school average, high school percentile rank, SAT math, SAT verbal, log of number of office hour visits, log of percent recitation usage, comfort level, Cornell critical thinking score, self-efficacy for self-regulation, self-efficacy for simple programming, attribution of success/failure on first exam to luck, percent of labs submitted, and number of exams taken. Descriptive statistics for these variables as well as the dependent variable are given in Table 5.15.

	Mean	Std. Deviation	*n*
Square root of Course Average	7.5576	2.0961	380
Credit Hours	15.0251	2.9919	499
HS Avg	89.1198	5.8651	253
HS Percentile Rank	75.7709	17.3053	199
SAT Math	602.3547	87.3588	327
SAT Verbal	550.4587	95.0659	327
Log of Number of Office Hour Visits	.5887	.8378	499
Log of Percent Recitation Usage	3.6901	1.2137	499
Comfort Level	3.2789	.5953	250
Cornell CT Score	13.7801	8.1296	216
SE - Self-regulation	3.7470	1.5752	250
SE - Simple Programming	4.7276	1.8937	250

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Att - Luck	2.0643	1.1482	249
Percent Labs Submitted	77.3951	28.8684	378
Num Exams Taken	2.6111	.8173	378

Table 5.15 Descriptive Statistics for Independent and Dependent Variables of Stepwise Multiple Linear Regression of Course Average

After these variables were identified, a stepwise multiple linear regression was run and further test assumptions were inspected. Tests of normality of the residuals and multicollinearity showed that neither was problematic. However, the test of homoscedasticity revealed a non-constant variance of the residuals by predicted value. The need for a transformation of the dependent variable became obvious. Log and square root transforms of the dependent variable were attempted to correct the problem of increasing variance. The square root transform was the most successful. The previously identified variables remained correlated with the square root transformed dependent variable. Further, no additional variables were correlated with the transformed dependent variable.

A stepwise multiple linear regression was conducted to determine the best model for the predictor variables (see Table 5.16 and Table 5.17).

					Change Statistics				
Model	R	R Square	Adjusted R Square	Std. Error of the Estimate	R Square Change	F Change	df1	df2	Sig. F Change
1	.868(a)	.754	.751	1.04560	.754	320.992	1	105	.00000
2	.931(b)	.866	.863	.77498	.112	87.135	1	104	.00000
3	.980(c)	.960	.959	.42461	.094	243.444	1	103	.00000
4	.985(d)	.969	.968	.37463	.009	30.314	1	102	.00000

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

5	.989(e)	.977	.976	.32274	.008	36.436	1	101	.00000
6	.991(f)	.981	.980	.29510	.004	20.812	1	100	.00001
7	.992(g)	.985	.983	.26991	.003	20.531	1	99	.00002
8	.995(h)	.991	.990	.20804	.006	68.637	1	98	.00000
9	.997(i)	.994	.993	.16961	.003	50.448	1	97	.00000
10	.997(j)	.994	.994	.16441	.000	7.231	1	96	.00845

a Predictors: (Constant), Percent Labs Submitted
b Predictors: (Constant), Percent Labs Submitted, Num Exams Taken
c Predictors: (Constant), Percent Labs Submitted, Num Exams Taken, Comfort Level
d Predictors: (Constant), Percent Labs Submitted, Num Exams Taken, Comfort Level, Cornell CT Score
e Predictors: (Constant), Percent Labs Submitted, Num Exams Taken, Comfort Level, Cornell CT Score, Att - Luck
f Predictors: (Constant), Percent Labs Submitted, Num Exams Taken, Comfort Level, Cornell CT Score, Att - Luck, Log of Percent Recitation Usage
g Predictors: (Constant), Percent Labs Submitted, Num Exams Taken, Comfort Level, Cornell CT Score, Att - Luck, Log of Percent Recitation Usage, SAT Math
h Predictors: (Constant), Percent Labs Submitted, Num Exams Taken, Comfort Level, Cornell CT Score, Att - Luck, Log of Percent Recitation Usage, SAT Math, HS Avg
i Predictors: (Constant), Percent Labs Submitted, Num Exams Taken, Comfort Level, Cornell CT Score, Att - Luck, Log of Percent Recitation Usage, SAT Math, HS Avg, SAT Verbal
j Predictors: (Constant), Percent Labs Submitted, Num Exams Taken, Comfort Level, Cornell CT Score, Att - Luck, Log of Percent Recitation Usage, SAT Math, HS Avg, SAT Verbal, Log of Number of Office Hour Visits

Table 5.16 Stepwise Multiple Linear Regression Models for Course Average

It is encouraging that the final model accounts for 99.4% of the variance in course averages, $F(10, 96) = 1713.351$, $p < .00001$. The reader should note that while the final model is rather complex with a total of ten predictor (see model 10 in Table 5.16), variables the first three variables of percent of labs submitted, number of exams taken, and comfort level alone account for 95.9% of the variance. The additional 7 variables only contribute an additional 3.5% of the predictive power of the model.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Model		Unstandardized Coefficients		Standardized Coefficients		
		B	Std. Error	β	*t*	*p*
1	(Constant)	2.680	.290		9.226	.00000
	Percent Labs Submitted	.063	.004	.868	17.916	.00000
2	(Constant)	1.382	.256		5.392	.00000
	Percent Labs Submitted	.039	.004	.541	10.766	.00000
	Num Exams Taken	1.202	.129	.469	9.335	.00000
3	(Constant)	-2.044	.261		-7.842	.00000
	Percent Labs Submitted	.032	.002	.437	15.434	.00000
	Num Exams Taken	1.338	.071	.522	18.820	.00000
	Comfort Level	1.114	.071	.316	15.603	.00000
4	(Constant)	-2.188	.231		-9.456	.00000
	Percent Labs Submitted	.032	.002	.436	17.475	.00000
	Num Exams Taken	1.337	.063	.521	21.324	.00000
	Comfort Level	1.054	.064	.299	16.485	.00000
	Cornell CT Score	.025	.005	.097	5.506	.00000
5	(Constant)	-1.656	.218		-7.598	.00000
	Percent Labs Submitted	.031	.002	.423	19.537	.00000
	Num Exams Taken	1.371	.054	.535	25.241	.00000
	Comfort Level	1.000	.056	.284	17.922	.00000

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

	Cornell CT Score	.024	.004	.093	6.081	.00000
	Att - Luck	-.169	.028	-.093	-6.036	.00000
6	(Constant)	-1.753	.200		-8.747	.00000
	Percent Labs Submitted	.028	.002	.392	18.741	.00000
	Num Exams Taken	1.315	.051	.513	25.712	.00000
	Comfort Level	.971	.051	.276	18.866	.00000
	Cornell CT Score	.024	.004	.093	6.687	.00000
	Att - Luck	-.169	.026	-.093	-6.603	.00000
	Log of Percent Recitation Usage	.139	.030	.080	4.562	.00001
7	(Constant)	-2.460	.241		-10.220	.00000
	Percent Labs Submitted	.028	.001	.379	19.633	.00000
	Num Exams Taken	1.310	.047	.511	28.001	.00000
	Comfort Level	.947	.047	.269	20.007	.00000
	Cornell CT Score	.018	.004	.070	5.061	.00000
	Att - Luck	-.176	.024	-.097	-7.495	.00000
	Log of Percent Recitation Usage	.152	.028	.088	5.445	.00000
	SAT Math	.002	.000	.063	4.531	.00002
8	(Constant)	-.249	.325		-.767	.44486
	Percent Labs Submitted	.028	.001	.382	25.644	.00000
	Num Exams Taken	1.358	.037	.529	37.176	.00000
	Comfort Level	.997	.037	.283	26.966	.00000

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

	Cornell CT Score	.015	.003	.058	5.389	.00000
	Att - Luck	-.197	.018	-.108	-10.750	.00000
	Log of Percent Recitation Usage	.159	.022	.092	7.368	.00000
	SAT Math	.003	.000	.111	9.098	.00000
	HS Avg	-.035	.004	-.099	-8.285	.00000
9	(Constant)	-.178	.265		-.672	.50301
	Percent Labs Submitted	.028	.001	.391	32.046	.00000
	Num Exams Taken	1.353	.030	.528	45.439	.00000
	Comfort Level	1.006	.030	.286	33.327	.00000
	Cornell CT Score	.009	.002	.034	3.630	.00045
	Att - Luck	-.207	.015	-.113	-13.814	.00000
	Log of Percent Recitation Usage	.160	.018	.093	9.114	.00000
	SAT Math	.002	.000	.088	8.423	.00000
	HS Avg	-.042	.004	-.117	-11.665	.00000
	SAT Verbal	.002	.000	.073	7.103	.00000
10	(Constant)	-.169	.257		-.656	.51342
	Percent Labs Submitted	.029	.001	.395	33.140	.00000
	Num Exams Taken	1.353	.029	.528	46.880	.00000
	Comfort Level	1.018	.030	.289	34.401	.00000
	Cornell CT Score	.009	.002	.033	3.688	.00037
	Att - Luck	-.211	.015	-.115	-14.442	.00000
	Log of Percent Recitation Usage	.172	.018	.100	9.777	.00000

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

SAT Math	.002	.000	.084	8.184	.00000
HS Avg	-.042	.003	-.118	-12.076	.00000
SAT Verbal	.002	.000	.073	7.351	.00000
Log of Number of Office Hour Visits	-.059	.022	-.024	-2.689	.00845

Table 5.17 Coefficients for Stepwise Multiple Linear Regression Models of Course Average

The inclusion of percent of labs submitted and number of exams taken was to investigate the effect of student effort on success. However these variables are "yoked" to course average, that is, course average is based in a large part on exam and lab averages. Since these variables are yoked to the course average the regression was run again without their inclusion. However, without the "yoked" variables the assumption of normality of distribution of the residuals was not met. Therefore the analysis was run with the non-transformed course average variable. This model showed no violation of the assumptions for the multiple linear regression test. Table5. 5.18 shows the model summary information without inclusion of the yoked variables, $F(3, 103) = 40.717$, $p < .00001$.

					Change Statistics				
Model	R	R Square	Adjusted R Square	Std. Error of the Estimate	R Square Change	F Change	df1	df2	Sig. F Change
1	.600(a)	.361	.354	20.432	.361	59.210	1	105	.00000
2	.695(b)	.483	.473	18.456	.123	24.680	1	104	.00000
3	.737(c)	.543	.529	17.449	.059	13.355	1	103	.00041

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

a Predictors: (Constant), Log of Percent Recitation Usage
b Predictors: (Constant), Log of Percent Recitation Usage, Comfort Level
c Predictors: (Constant), Log of Percent Recitation Usage, Comfort Level, SAT Math

Table5. 5.18 Stepwise Linear Regression Models of Course Average sans Yoked Variables

Table 5.19 shows the coefficients for the models. It should be noted that the predictor variables revealed are the same as those for the transformed dependent variable i.e., square root of course average, without the yoked variables.

Model		Unstandardized Coefficients		Standardized Coefficients		
		B	Std. Error	β	*t*	*p*
1	(Constant)	15.072	6.349		2.374	.01941
	Log of Percent Recitation Usage	12.582	1.635	.600	7.695	.00000
2	(Constant)	-29.439	10.638		-2.767	.00669
	Log of Percent Recitation Usage	11.072	1.508	.528	7.342	.00000
	Comfort Level	15.274	3.075	.358	4.968	.00000
3	(Constant)	-66.623	14.307		-4.657	.00001
	Log of Percent Recitation Usage	11.050	1.426	.527	7.751	.00000
	Comfort Level	13.404	2.952	.314	4.541	.00002
	SAT Math	.072	.020	.247	3.654	.00041

Table 5.19 Coefficients for Stepwise Multiple Linear Regression of Course Average sans Yoked Variables

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

5.1.6.2 Lab Average

Prior to the multiple regression, scatterplots and Pearson correlation coefficients were computed for each of the independent variables, with lab average to guard the assumption of linear regression that a linear relationship does in fact exist between the each independent variable and the dependent variable (see Appendix D).

An examination of the scatterplots once again revealed the need for logarithmic transformation of the number of office hours visits and percent recitation utilization variables. The function $ln(1 + x)$ was applied for each of these variables for the current analysis. Only those variables that revealed significant values for Pearson correlation coefficients were used in building the model for lab average. These variables are: number of credit hours taken during the current semester, high school average, high school percentile rank, SAT math score, log of number of office hour visits, log of the percent recitation utilization, comfort level, Cornell critical thinking score, self-efficacy self-regulation dimension, and attribution of success/failure to luck. Table 5.20 shows the descriptive statistics for the independent and dependent variables.

	Mean	Std. Deviation	n
Lab Avg	62.5978	28.7260	380
Credit Hours	15.0251	2.9919	499
HS Avg	89.1198	5.8651	253
HS Percentile Rank	75.7709	17.3053	199
SAT Math	602.3547	87.3588	327

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Log of Number of Office Hour Visits	.5887	.8378	499
Log of Percent Recitation Usage	3.6901	1.2137	499
Comfort Level	3.2789	.5953	250
Cornell CT Score	13.7801	8.1296	216
SE - Self-regulation	3.7470	1.5752	250
Att - Luck	2.0643	1.1482	249

Table 5.20 Descriptive Statistics for Stepwise Multiple Linear Regression for Lab Average

After these variables were identified, a stepwise multiple linear regression was run and further test assumptions were inspected. There were no violations of test assumptions. Table 5.21 shows amount of variance accounted for by each of the models.

					Change Statistics				
Model	R	R Square	Adjusted R Square	Std. Error of the Estimate	R Square Change	F Change	df1	df2	Sig. F Change
1	.583(a)	.340	.334	23.450	.340	54.063	1	105	.00000
2	.676(b)	.457	.446	21.373	.117	22.404	1	104	.00001
3	.695(c)	.483	.468	20.960	.026	5.135	1	103	.02553
4	.715(d)	.511	.492	20.476	.028	5.929	1	102	.01663

a Predictors: (Constant), Log of Percent Recitation Usage
b Predictors: (Constant), Log of Percent Recitation Usage, Comfort Level
c Predictors: (Constant), Log of Percent Recitation Usage, Comfort Level, SAT Math
d Predictors: (Constant), Log of Percent Recitation Usage, Comfort Level, SAT Math, Log of Number of Office Hour Visits

Table 5.21 Stepwise Multiple Linear Regression for Lab Average

Log of percent recitation usage, comfort level, SAT math, and log of the number of office hour visits account for 49.2% of the variance in lab average scores. This model is

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

statistically significant, $F(4, 102) = 26.658$, $p < .00001$. Table 5.22 shows the coefficients for each of the models.

Model		Unstandardized Coefficients		Standardized Coefficients		
		B	Std. Error	β	*t*	*p*
1	(Constant)	11.681	7.286		1.603	.11192
	Log of Percent Recitation Usage	13.798	1.877	.583	7.353	.00000
2	(Constant)	-37.430	12.319		-3.038	.00301
	Log of Percent Recitation Usage	12.133	1.746	.513	6.948	.00000
	Comfort Level	16.852	3.560	.349	4.733	.00001
3	(Constant)	-65.127	17.185		-3.790	.00025
	Log of Percent Recitation Usage	12.116	1.712	.512	7.075	.00000
	Comfort Level	15.459	3.545	.320	4.360	.00003
	SAT Math	.054	.024	.163	2.266	.02553
4	(Constant)	-63.778	16.797		-3.797	.00025
	Log of Percent Recitation Usage	10.355	1.823	.438	5.682	.00000
	Comfort Level	13.933	3.520	.289	3.959	.00014
	SAT Math	.064	.024	.195	2.725	.00756

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Log of Number of Office Hour Visits	6.539	2.686	.191	2.435	.01663

Table 5.22 Coefficients for Independent and Dependent Variables of Stepwise Multiple Linear Regression Models of Lab Average

5.1.6.3 Exam Average

Prior to the multiple regression, scatterplots and Pearson correlation coefficients were computed for each of the independent variables, with exam average to guard the assumption of linear regression that a linear relationship does in fact exist between the each independent variable and the dependent variable (see Appendix E).

An examination of the scatterplots once again revealed the need for logarithmic transformation for the number of office hours visits and percent recitation utilization variables. The function $ln(1 + x)$ was applied for each of these variables for the current analysis. Only those variables that revealed significant values for Pearson correlation coefficients were used in building the model for exam average. These variables are: number of credit hours taken during the current semester, high school average, high school percentile rank, SAT math score, SAT verbal score, log of the number of office hour visits, log of percent recitation utilization, comfort level, Cornell critical thinking score, self-efficacy self-regulation dimension, self-efficacy simple programming dimension, attribution for success/failure to luck, log of the total number of years of prior programming experience, log of the number of programming language known, and the number of labs submitted.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

	Mean	Std. Deviation	*n*
Exam Avg	56.8001	25.6330	380
Credit Hours	15.0251	2.9919	499
HS Avg	89.1198	5.8651	253
HS Percentile Rank	75.7709	17.3053	199
SAT Math	602.3547	87.3588	327
SAT Verbal	550.4587	95.0659	327
Log of Number of Office Hour Visits	.5887	.8378	499
Log of Percent Recitation Usage	3.6901	1.2137	499
Comfort Level	3.2789	.5953	250
Cornell CT Score	13.7801	8.1296	216
SE - Self-regulation	3.7470	1.5752	250
SE - Simple Programming	4.7276	1.8937	250
Att - Luck	2.0643	1.1482	249
Log of Total Year Programming	1.1080	.8500	264
Log of Number of Programming Languages	.9022	.5371	264
Percent Labs Submitted	77.3951	28.8684	378

Table 5.23 Descriptive Statistics for Independent and Dependent Variables of Stepwise Multiple Linear Regression for Exam Average

After these variables were identified, a stepwise multiple linear regression was run and further test assumptions were inspected. There were no violations of test assumptions. Table 5.24 shows the proportion of variance accounted for by each of the models.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Model	R	R Square	Adjusted R Square	Std. Error of the Estimate	Change Statistics				
					R Square Change	F Change	df1	df2	Sig. F Change
1	.718(a)	.516	.511	17.9205	.516	111.872	1	105	.00000
2	.768(b)	.590	.582	16.5720	.074	18.784	1	104	.00003
3	.788(c)	.622	.611	15.9968	.032	8.614	1	103	.00411

a Predictors: (Constant), Percent Labs Submitted
b Predictors: (Constant), Percent Labs Submitted, Comfort Level
c Predictors: (Constant), Percent Labs Submitted, Comfort Level, SAT Verbal

Table 5.24 Stepwise Multiple Linear Regression Models for Exam Average

Percent of labs submitted, comfort level and SAT verbal scores account for 61.1% of the variance of exam average scores, $F(3, 103) = 56.390$, $p < .00001$. Table 5.25 shows the coefficients for each of the models.

Model		Unstandardized Coefficients		Standardized Coefficients		
		B	Std. Error	β	*t*	*p*
1	(Constant)	7.443	4.978		1.495	.13784
	Percent Labs Submitted	.638	.060	.718	10.577	.00000
2	(Constant)	-27.833	9.351		-2.977	.00363
	Percent Labs Submitted	.586	.057	.660	10.266	.00000
	Comfort Level	11.989	2.766	.278	4.334	.00003
3	(Constant)	-51.850	12.183		-4.256	.00005
	Percent Labs Submitted	.587	.055	.661	10.663	.00000
	Comfort Level	11.181	2.684	.260	4.165	.00006

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

SAT Verbal	.048	.016	.179	2.935	.00411

Table 5.25 Coefficients for Stepwise Multiple Linear Regression Models of Exam Average

5.2 Resignations

5.2.1 By Gender

In order to explore the idea of the effect of gender on resign rate a Chi-square (χ^2) test of association was performed (see Table 5.26). The test failed to reveal gender bias in resign rate ($\chi^2(1) = .853$, $p = .35835$).

			Resigned		Total
			No	Yes	
GENDER	F	Count	47	19	66
		Expected Count	50.0	16.0	66.0
		% within GENDER	71.2%	28.8%	100.0%
		Std. Residual	-.4	.7	
	M	Count	330	102	432
		Expected Count	327.0	105.0	432.0
		% within GENDER	76.4%	23.6%	100.0%
		Std. Residual	.2	-.3	
Total		Count	377	121	498
		Expected Count	377.0	121.0	498.0
		% within GENDER	75.7%	24.3%	100.0%

Table 5.26 Gender by Resign Rate

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

5.2.2 By Credit Hours

To test whether resigns might be due to students' overestimating the work they could get done in a semester, an independent samples t-test was planned. However, the number of credit hours were non-normally distributed by whether students resigned, therefore the non-parametric, Mann-Whitney U was run. The test indicated a significant effect (see Table 5.27).

	Credit Hours
Mann-Whitney U	19400.000
Wilcoxon W	90653.000
z	-2.501
p	.01239

Table 5.27 Mann-Whitney U test for Number of Credit Hours by Resign

Students who resigned were registered for more credit hours at the beginning of the semester than those who did not.

5.2.3 By Year in School

The effect of year in school on resign rate was assessed through a Chi-square test of association. Masters and doctoral students were once again collapsed into a single graduate (GR) category as in section 5.1.3. Table 5.28 shows a cross-tabulation of year in school by resign rate. The Chi-square test failed to reveal any degree of association between year in school and resign rate ($\chi^2(5) = 3.116$, $p = .68205$). Conservative statisticians may be alarmed by the fact that three cells (25%) have expected counts less

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

than 5 and that one cell has an expected count < 1. Although the issue of expected counts being too small is contentious (Howell, 1999), the analysis was also run without the problematic graduate (GR) group. The latter analysis likewise failed to show a statistically significant association ($\chi^2(4) = 2.139$, $p = .71023$).

			Resigned		Total
			No	Yes	
Year in School	FR	Count	193	71	264
		Expected Count	199.9	64.1	264.0
		% within Year in School (scalar)	73.1%	26.9%	100.0%
		Std. Residual	-.5	.9	
	SO	Count	70	19	89
		Expected Count	67.4	21.6	89.0
		% within Year in School (scalar)	78.7%	21.3%	100.0%
		Std. Residual	.3	-.6	
	JR	Count	60	15	75
		Expected Count	56.8	18.2	75.0
		% within Year in School (scalar)	80.0%	20.0%	100.0%
		Std. Residual	.4	-.8	
	SR	Count	41	13	54
		Expected Count	40.9	13.1	54.0

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

		% within Year in School (scalar)	75.9%	24.1%	100.0%
		Std. Residual	.0	.0	
	GR	Count	3	0	3
		Expected Count	2.3	.7	3.0
		% within Year in School (scalar)	100.0%	.0%	100.0%
		Std. Residual	.5	-.9	
	NM	Count	10	3	13
		Expected Count	9.8	3.2	13.0
		% within Year in School (scalar)	76.9%	23.1%	100.0%
		Std. Residual	.1	-.1	
Total		Count	377	121	498
		Expected Count	377.0	121.0	498.0
		% within Year in School (scalar)	75.7%	24.3%	100.0%

Table 5.28 Year in School by Resign Rate

5.2.4 By Major

In an attempt to answer the research question: What is the effect of major/intended major on resign rates? Refer to Table 5.9 above for the major abbreviations.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

A Chi-square test of association was performed to determine if rate of resignation was higher for any of the individual majors. Unfortunately, 43 cells (77%) have expected counts lower than 5, thus violating the assumptions of the Chi-square test (see Table 5.29).

			Resigned		Total
			No	Yes	
Major	ARC	Count	1	0	1
		Expected Count	.8	.2	1.0
	ASE	Count	4	2	6
		Expected Count	4.6	1.4	6.0
	BCH	Count	1	1	2
		Expected Count	1.5	.5	2.0
	BIO	Count	4	0	4
		Expected Count	3.0	1.0	4.0
	CE	Count	0	1	1
		Expected Count	.8	.2	1.0
	CEN	Count	84	25	109
		Expected Count	82.8	26.2	109.0
	CHE	Count	3	2	5
		Expected Count	3.8	1.2	5.0
	CIE	Count	2	0	2
		Expected Count	1.5	.5	2.0
	CPP	Count	2	0	2
		Expected Count	1.5	.5	2.0
	CS	Count	191	41	232

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

	Expected Count	176.1	55.9	232.0
DAC	Count	2	0	2
	Expected Count	1.5	.5	2.0
DMS	Count	7	2	9
	Expected Count	6.8	2.2	9.0
EAS	Count	7	2	9
	Expected Count	6.8	2.2	9.0
ECO	Count	1	0	1
	Expected Count	.8	.2	1.0
EE	Count	5	7	12
	Expected Count	9.1	2.9	12.0
ENG	Count	4	3	7
	Expected Count	5.3	1.7	7.0
FA	Count	3	0	3
	Expected Count	2.3	.7	3.0
GEO	Count	0	1	1
	Expected Count	.8	.2	1.0
HIS	Count	0	1	1
	Expected Count	.8	.2	1.0
ME	Count	4	1	5
	Expected Count	3.8	1.2	5.0
MG	Count	8	8	16
	Expected Count	12.1	3.9	16.0
MGA	Count	2	2	4
	Expected Count	3.0	1.0	4.0
MTH	Count	3	6	9

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

		Expected Count	6.8	2.2	9.0
	PHY	Count	1	0	1
		Expected Count	.8	.2	1.0
	PSY	Count	7	2	9
		Expected Count	6.8	2.2	9.0
	SSC	Count	3	1	4
		Expected Count	3.0	1.0	4.0
	TH	Count	1	0	1
		Expected Count	.8	.2	1.0
	UNM	Count	19	9	28
		Expected Count	21.3	6.7	28.0
Total		Count	369	117	486
		Expected Count	369.0	117.0	486.0

Table 5.29 Major by Resign Rate

Therefore, the majors/intended major of the students were reclassified based on type. The reclassification was the same as described in section 5.1.4 (refer to Table 5.10 above for the mapping).

A Chi-square analysis was carried out on the Type of Major/Intended Major, see Table 5.30 for the new cross-tabulation.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

			Resigned		Total
			No	Yes	
Major (by Type)	ART	Count	13	2	15
		Expected Count	11.4	3.6	15.0
		% within Major (by Type)	86.7%	13.3%	100.0%
		Std. Residual	.5	-.8	
	BUS	Count	11	10	21
		Expected Count	15.9	5.1	21.0
		% within Major (by Type)	52.4%	47.6%	100.0%
		Std. Residual	-1.2	2.2	
	CEN	Count	84	25	109
		Expected Count	82.8	26.2	109.0
		% within Major (by Type)	77.1%	22.9%	100.0%
		Std. Residual	.1	-.2	
	CS	Count	191	41	232
		Expected Count	176.1	55.9	232.0
		% within Major (by Type)	82.3%	17.7%	100.0%
		Std. Residual	1.1	-2.0	
	ENG	Count	23	13	36
		Expected Count	27.3	8.7	36.0
		% within Major (by Type)	63.9%	36.1%	100.0%

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

		Std. Residual	-.8	1.5	
	HUM	Count	4	4	8
		Expected Count	6.1	1.9	8.0
		% within Major (by Type)	50.0%	50.0%	100.0%
		Std. Residual	-.8	1.5	
	MTH	Count	3	6	9
		Expected Count	6.8	2.2	9.0
		% within Major (by Type)	33.3%	66.7%	100.0%
		Std. Residual	-1.5	2.6	
	SCI	Count	21	7	28
		Expected Count	21.3	6.7	28.0
		% within Major (by Type)	75.0%	25.0%	100.0%
		Std. Residual	-.1	.1	
	UNM	Count	19	9	28
		Expected Count	21.3	6.7	28.0
		% within Major (by Type)	67.9%	32.1%	100.0%
		Std. Residual	-.5	.9	
Total		Count	369	117	486
		Expected Count	369.0	117.0	486.0
		% within Major (by Type)	75.9%	24.1%	100.0%

Table 5.30 Type of Major/Intended Major by Resign Rate

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

The results reveal a statistically significant difference for resignation rates, $\chi^2(8) = 28.404$, $p = .00041$. Post-hoc binomial tests revealed that Business (BUS) and Math (MTH) had statistically significantly more resignation than expected ($p = .031$ and $p = .016$, respectively), while Computer Science (CS) had fewer ($p = .025$). However, the remaining categories showed no significant departure from expected frequencies.

5.3 Summary

This chapter has reported a lack of gender bias for both success and resignation. Major/intended major as well as reason for computer science major/intended major likewise failed to show differences for success.

Stepwise multiple linear regressions for course average showed high predictive ability (> 90%) with work completed and comfort level being the greatest predictors. A secondary multiple linear regression for course average revealed time spent in recitation and comfort level as the main predictors along with a small effect for SAT math scores. The stepwise multiple linear regression for lab average upholds recitation attendance and comfort level as primary predictors with SAT math and number of office hour visits adding small support. Exam average was predicted by a stepwise multiple linear regression model consisting of percent of labs submitted, comfort level, and SAT verbal scores.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

In addition to lack of gender bias, year in school revealed no influence on resignation. Business and mathematics majors resigned the course more than expected while computer science students had lower than expected resignation rates.

The next chapter will discuss these findings in the larger CS1 and CS education contexts with special emphasis on the differences between predictors for traditional CS1 courses and those for the OF CS1.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Chapter 6

Discussion of Experimental Results

This chapter discusses the experimental results reported in the last chapter with an emphasis on the focus of this dissertation namely, traditional versus OF CS1, as well as the larger context of computer science education.

6.1 Lack of Gender Bias for Success and Resignations

With concern among the computer science education community about the lack of women in computer science (Cuny & Aspray, 2001; Margolis & Fisher, 2002; Roberts, Kassianidou, & Irani, 2002), any analysis of new techniques for CS1 would be remiss without considering gender bias. CS1 enrollments generally show females outnumbered by their male peers; the current study supports this notion with only 66 (13.2%) female students to 433 (86.8%) males. This discrepancy makes it crucial that CS1 instructors not alienate women early on, when diversity and gender equity are the goals. It is therefore

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

encouraging to see that gender is not a factor for success or resignation rates in the OF CS1.

6.2 Success

This section discusses the independent findings for prior programming experience, year in school, major/intended major, reason for CS major/intended major, as well as, the multivariate models of success.

6.2.1 Prior Programming Experience

As we have seen the analysis regarding prior programming experience as a predictor of success failed to show that students with prior programming experience performed better in any of the measures of success.

Given the focus in the OF CS1 on design aspects and to a lesser degree object-oriented programming, it is not surprising that those students with prior programming experience (in general) do no better than their non-programming peers. OO is a unique way of thinking about problem solving (as are all the major language paradigms) and being trained in other paradigms provides no added benefit.

The finding that prior C++ and Java experience provide no benefit along with the finding that prior Java experience detracts from performance on the exam is, at first look, surprising. However, generally C++ and Java books and the courses that use them are largely imperative in nature. They do not teach students to embrace OO as a unique way

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

of thinking. This conclusion seems to be supported by the relation of prior Java experience and exam average. The exams test OO concepts including modeling using class diagrams, modification of program design at the class diagram level, application of design patterns, and writing small amounts of Java code all in a limited time frame. It is believed that the prior imperative Java experience that students come in with results in a *cognitive dissonance* about Java. That is, students cannot resolve the two radically different approaches to using Java in the limited time for the exams.

Recall that there was no correlation between the measures of success and student self-efficacy for object-orientation. There are two possible explanations for the lack of correlation. First, despite the fact that the self-efficacy test was given after many weeks of object-oriented instruction, students' own notions of what OOP is may still differ from reality. While somewhat plausible, this explanation is suspect, given that they had already seen the core of OOP. Anecdotally, a number of students comment very early on in the semester that OOP is very different from what they thought it to be.

Given that it is likely that students at the time of taking the survey do know what OOP is coupled with the results of prior programming experience, both general and OO languages, it is reasonable to believe the lack of correlation is due to the difference in the OF CS1 curriculum. That is, prior programming experience is not a predictor of success for the objects-first CS1.

It is believed that object-oriented design is a much more natural way of problem solving and that this leads to the lack of effect for prior programming experience.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

The fact that measures of success did not show significant differences for prior programming experience "flies in the face" of past research (Cantwell Wilson & Shrock, 2001; Evans & Simkin, 1989; Hagan & Markham, 2000), as well as professional belief (Curricula, 2001). This finding has important ramifications for CS1 educators. That is, given the disparate backgrounds of students with regard to prior programming experience, it is helpful to have a CS1 that does not divide the students along such lines.

6.2.2 Year in School

The investigation of year in school failed to show differences for all categories; i.e., no pattern could be discerned from the data. However, there were differences between the lab average scores for sophomores and seniors; namely, seniors performed better on the labs than their sophomore counterparts. An examination of the predictors of success for lab average, namely, percent recitation utilization, comfort level, SAT math, and number of office hour visits, failed to reveal any differences by year in school.

The tests for exam and course averages were close to statistical significance with $p = .05931$ and $p = .06300$, respectively. It would seem that once again this is due to sophomore versus senior differences. These differences may be due in a small part to "sophomore slump." However, the lack of significance in the differences between sophomores and all others suggests that the effect of sophomore slump is minimal at best.

In an attempt to explain the sophomore/senior differences, an analysis of the grade point averages (GPA) of sophomores and seniors was undertaken. A Mann-Whitney U

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

test showed a significant difference between sophomore (n = 87) and senior (n = 54) GPAs (z = -2.676, p = .00744). Therefore it appears that the sophomore/senior differences are mainly due to differences in GPA.

6.2.3 Major/Intended Major

The analysis of major/intended major failed to show an effect for success across any of the measures of success. This finding is encouraging, as it suggests that the OF CS1 course is "at the right level." That is, it does not discriminate against non-major students.

However there are a few groups that are underrepresented in the current analysis, namely humanities (1%) and math (.8%) students. The mean ranks for math students do not appear at either extreme of the ranks. In contrast, the humanities students consistently showed the lowest mean rank for each measure of success. The humanities group for this analysis consisted of the four English majors who did not resign the course. A closer look at their data revealed that none of these students took the final exam, and only one of them took the second in-class exam. Further, percent of labs submitted was less than 60% for three of the students – the missing labs were the later ones. It appears that the English majors by and large "gave up" in the course. Given that there were only four students in the humanities group, future research is needed to reveal whether this is a trend or an anomaly.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

6.2.4 Reason for CS Major/Intended Major

At the outset of this study, it was believed that reason for CS major would be a reasonable indicator of success. It was believed that students who took up the major due to parental influence, for the promise of a high paying job, or because they enjoyed playing computer games or "surfing the web" would do worse than those students who professed interest. This belief was based largely on the findings of (Sheard & Hagan, 1998), who reported that students repeating an introductory programming course had "much less desire for learning computing and in particular programming." The data in the current study did not bear out these hypotheses. The "highest" performers in terms of mean ranks were the two students who choose computer science as a major because their parents told them to and the three students who were advised to take up computer science by their high school guidance counselors. However, the low numbers in these groups call for further studies. It may be that such reasons would play a bigger role in resignation rates.

6.2.5 Multivariate Models of Course Average

This section discusses multivariate models of course average. Both full and non-yoked models are presented.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

6.2.5.1 Full Models of Course Average

With regard to course average the full model shows (in order of inclusion in the model), percent of labs submitted, number of exams taken, comfort level, Cornell critical thinking score, attribution of success/failure to luck, (natural log of) percent recitation usage, SAT math score, high school average, SAT verbal score, and (natural log of) number of office hour visits as predictors. While this ten predictor model maximizes predictive power, the simpler seven variable model that excludes high school average, SAT verbal, and number of office hour visits will be the focus of this discussion. The seven variable model accounts for 98.3% of the variance of course average (a loss of only 1.1% of the variance). This is the first model identified by the stepwise multiple linear regression to introduce SAT math scores.

It is interesting to note that three out of seven predictors are directly related to the effort a student puts into the course, namely percent of labs submitted, number of exams taken, and percent recitation utilization. Percent of labs submitted and number of exams taken were the first variables entered into the models, and account for 86.3% of the variance in course scores. The last variable, percent recitation utilization, has a stronger effect than SAT math scores.

Psychological factors such as comfort level ($\beta = .269$) and attribution of success/failure to luck ($\beta = -.097$) were also more predictive than SAT math scores ($\beta = .063$).

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Cornell critical thinking scores were fourth in introduction to the models and represented the first addition of an "academic" predictor. These data may be explained by the work of (Steele Hanson, 1986). Steele Hanson found differences in the critical thinking ability of novice versus expert programmers, namely expert programmers showed greater Cornell Critical Thinking scores.

SAT math scores were last entered into the seven variable model and only account for an additional .3% of the variance of scores. In short, while SAT math scores are correlated with course average ($r = .267$, $p = .00001$), in the presence of the other six predictors, the value of knowing SAT math scores is negligible.

6.2.5.2 Non-Yoked Model of Course Average

The final model for course average sans percent of labs submitted and number of exams taken revealed three predictors of success. Percent of recitation usage replaced percent of labs submitted and number of exams taken as the dominant predictor. This was followed by comfort level and finally SAT math scores. Once again, it is telling to see that effort and comfort level have a greater impact on success than do SAT math scores. In fact the SAT math scores only add an additional 5.9% of predictive power to the model. The non-yoked model revealed the same pattern of effort, followed by comfort level, followed by academic variables revealed in the full model.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

6.2.6 Multivariate Model of Lab Average

The model for lab average indicated percent recitation usage, comfort level, SAT math score, and number of office hour visits as predictors. Once again effort in terms of percent of recitation utilization, and comfort level, provide the greatest predictive power, accounting for 44.6% of the variance in lab average. SAT math scores only contribute 2.6% of the variance.

6.2.7 Multivariate Model of Exam Average

Exam averages are best predicted by the three variable model consisting of percent of labs submitted, comfort level, and SAT verbal scores. Percent of labs submitted together with comfort level account for 58.2% of the variance of exam averages. Once again SAT scores, this time SAT verbal, contribute very little to the predictive power (2.9%).

6.2.8 Summary of Multivariate Models of Success

For every measure of success in this study, i.e., course, lab, and exam averages, measures of effort consistently showed the highest predictive value. That is, if one had to choose only a single factor, it should be effort. These analyses support the idea that students get out of the course what they put into it. This fact alone goes a long way to dispel the myth that OOP and OOD are too difficult for a CS1 course.

The fact that comfort level is an important factor is supported by the work of (Cantwell Wilson & Shrock, 2001). Despite the findings that student effort plays a

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

central role, it is important as educators that we try to maximize the degree to which students feel comfortable in the class. For CS1 courses, this includes not only comfort level with programming and non-programming computer use, but also comfort with the instructional staff. That is, students need to be encouraged to attend office hours, and made to feel that their questions in class are welcomed and valued.

The consistent finding that academic predictors such as critical thinking ability as well as SAT scores repeatedly turned up last, adding only small predictive value, is both surprising and encouraging. It is surprising in the sense that it contradicts the findings of previous work (see Predictors Background). It is encouraging in that it supports the conclusion that the OF CS1 is "at the right level," that is, not too difficult for introductory students.

6.3 Resignations

This section discusses the data on a handful of possible reasons for why students resign. It is crucial to remember that resignation does not equal failure. There are a number of reasons why students may resign the course, among them:

- Are they resigning to avoid a poor grade? If so, is it due to some lack of prerequisite knowledge or personal factors?
- Have they concluded that they are not really interested in the study of computer science after all?

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

- Did they find their workload was too high?

The last question was examined through analysis of the number of credit hours a student was registered for. Unfortunately, data collection on the resign group has proven to be difficult. The data on students who resigned is therefore quite limited. Despite efforts of this study to gather data early enough in the semester, these students did not engage in the study. This means that variables such as Cornell critical thinking score and comfort level were simply not available, due to non-participation. Furthermore, data on percent recitation utilization and percent recitation usage could not be computed, as the date on which resignation took place is not available.

6.3.1 Credit Hours

The finding that students who resign were registered for more credit hours than those who did not is not surprising. It does, however, begin to shed light on the OF CS1. The fact that students who resigned were registered for more credit hours hints at problems of time management as the reason for resignation as opposed to the OF CS1 being too difficult.

6.3.2 Year in School

The test for effect of year in school on resignation rate failed to reveal any differences. Despite the fact that sophomores showed (statistically) lower lab averages

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

than seniors, they showed no tendency to resign more than the other students. A careful look at the raw data shows that none of the groups were even close to showing a trend in the area of resignation.

6.3.3 Major/Intended Major

The analysis of major/intended major revealed differences in resignation rate for only three majors, computer science, business, and mathematics. Computer science students resigned less than the expected rate. One possible explanation for this trend is that CSE115 is a required course for computer science and intended computer science majors. However, CSE115 is also required of computer engineering majors, who did not show the same pattern of fewer resigns. While it is tempting to believe that this is the result of the course resonating with the computer science majors/intended majors there is another possible explanation. A Mann-Whitney U ($z = -5.18$, $p < .00001$) indicated that computer science majors ($n = 232$) were registered for fewer credit hours than computer engineering majors/intended majors ($n = 110$). In fact, as shown by another Mann-Whitney U ($z = -4.27$, $p = .00002$), computer science majors/intended majors ($n = 232$) registered for fewer credit hours than the other majors/intended majors ($n = 255$). Therefore it is likely that the reason for the low resign rate among computer science students is due to the (relatively) low number of credit hours they were registered for.

There were ten business students (47.6%) who resigned the course, eight were management majors and the remaining two were management-accounting majors. This

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

number was statistically significant. Given the importance of number of credit hours for resignation, the business students' number of credit hours was checked against that of the other majors, with no differences noted. An alternate explanation may be had in examining the reason these students take CSE115. The School of Management at University at Buffalo requires that their management information sciences (MIS) majors take either CSE113 and CSE114, or CSE115 and CSE116. Unfortunately, the major code for these students are only listed as the more generic management or management-accounting. CSE113 and CSE114 were designed to primarily serve this MIS audience. These classes are considerably different in course structure and ease of tasks. For instance, all exams in CSE113 and CSE114 are multiple choice while CSE115 and CSE116 exams are generally short-answer and coding/design. The labs in CSE113 are considerably shorter and provide substantially more help for the students. Many of the CSE113 labs simply ask the students to type in code from the lab handout. The labs do not exceed 50 lines of code. On the other hand, CSE115 students are allowed more freedom to make design choices, and the final lab is on the order of 1200 lines of code. It is believed that the workload differences coupled with the fact that management students can meet their requirements with CSE113 are the predominant reasons for the greater than expected resign rate among management students.

Of the ten business students who resigned, four students decided to take an introductory computer science course (either CSE113 or CSE115) again. One student took CSE113 in the Spring 2003 semester (following resigning CSE115 in the Fall 2002

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

semester) and received an A in CSE113. The remaining three students are enrolled in the Fall 2003 semester, two in CSE113 and one in CSE115. The intentions of the remaining six students are unclear. It is possible that they have simply decided to avoid computer science courses or take an introductory computer science course at a later date. The trend in CSE113 enrollment would seem to support this idea. Juniors made up the largest group of management students in CSE113, followed by sophomores.

Student mathematicians had the highest rates of resignations (66.7%). Prima facie it would appear that contrary to popular belief as well as prior research about the relationship between computer science and math, objects scare away mathematicians. However, there are likely some additional factors at play. Of the six math students who resigned, only one has a concentration that required CSE115. The remaining students either had no listed concentration or a concentration for which CSE115 was not a required course. The Mathematics Department at University at Buffalo recommends that undergraduate math students take either CSE113 and CSE114 (the non-majors introductory sequence) or CSE115 and CSE116. Their web page contains a strong recommendation for CSE115 and CSE116 over the non-majors sequence.

Considering the discrepancies regarding CSE113 and CSE115 described earlier it is possible that the students who resigned CSE115 did so in favor of the arguably much easier non-majors course sequence. However, when the enrollment data was examined it revealed that none of the math students either retook CSE115 or enrolled in CSE113.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

The analysis of credit hours may shed light on this matter. A look at the mean ranks of number of credit hours shows that mathematics students were registered for the greatest number of credit hours. While the difference between the mathematics students' number of credit hours and those of other majors is not significant, it was close (p = .05512). Therefore considering the relationship between number of credit hours a student was registered for and resign rates coupled with the somewhat high number of credit hours taken by the mathematics students, it is believed that math students' high resignations are largely are factor of their course load. Future research is still needed for more conclusive evidence.

6.4 Summary and Conclusions

6.4.1 How do predictors of success differ for OF CS1?

As was discussed in Chapter 2, imperatives-first CS1 courses are well studied. Prior studies have shown that mathematics is a predictor of success for imperatives-first courses. However, number of high school math courses were not predictive of success for the OF CS1. Additionally the predictive value of SAT math scores is slight.

Prior programming experience was also shown to be a predictor of success for imperatives-first courses, while such a relationship did not hold for the objects-first CS1.

While a number of cognitive, psychological, and behavioral variables were found to be important for imperatives-first introductory courses, only one is noteworthy. Cantwell

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Wilson and Shrock (2001) found comfort level was the best predictor of success for their course. The multivariate analyses carried out for the OF CS1 found that comfort level was the second strongest predictor of success for the course.

While other academic variables such as verbal ability were predictive for imperatives first CS1, they have very little predictive value for the OF CS1 course.

The primary predictor of success identified by the multivariate analyses of this dissertation is effort, which was not included as a variable in any of the prior research.

6.4.2 A CS1 for everyone

The University at Buffalo is unique in that it provides two different tracks of introductory computer science courses, one for majors and one for non-majors. At many institutions the CS1 instructors are teaching not only computer science majors but also the non-majors right alongside them.

The lack of evidence to show gender bias with regard to success or resign rate as well as the lack of influence of major/intended major for success in the OF CS1 suggest that the OF CS1 would be well suited in diverse liberal arts settings. This is supported by the finding that student effort and comfort level are the primary factors for all success. The fact that higher number of credit hours is associated with resignation suggests that the prime reason for resignation may be a factor of students "biting off more than they can chew."

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

An added benefit for the OF CS1 regardless of whether is it a liberal arts or majors only course is that prior programming experience is not a predictor of success. In the absence of prerequisite courses for CS1, instructors will be in the position of teaching students with diverse backgrounds with regard to programming experience. In fact CC2001 notes, "[p]rogramming-intensive courses disadvantage students who have no prior exposure to computers ... As a result, students who are new to computing are often overwhelmed." The data for the OF CS1 dispute this conjecture. This finding is significant in that the research on traditional courses consistently found prior programming experience to be a predictor.

While SAT math scores appeared as a predictor for two of the measures of course success, the conclusion that mathematical aptitude being important for the OF CS1 is suspect. As mentioned, the correlation between SAT math scores and course average is small ($r = .267, p = .00001$). This means that SAT math scores, alone, only account for 7.1% of the variance in course average. It is suspected that the academic predictors, namely critical thinking ability, and SAT scores, are predictive for success in the course to the degree which they are measures of general intelligence.

The fact that academic predictors in general, and SAT math scores in particular, have minimal predictive value for the multivariate models, casts considerable doubt on the idea that students need to be good at math to do well in CS1.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

What is clear is that the low predictive power of the academic variables, critical thinking, and SAT scores, coupled with the other data previously presented, converge on the conclusion that the OF CS1 is a CS1 for all everyone.

These data emphatically refute the assertion that OOP/OOD are too difficult to teach CS1 students. If OOP/OOD were difficult it is expected that academic variables would be the most predictive of success. To the contrary, academic predictors are the least powerful for the OF CS1.

While the findings are quite surprising given the past literature as well as long-held belief, the data collected spanned multiple class sections with multiple instructors and was gathered across three semesters. Furthermore, the large sample size provides for generous statistical power.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Chapter 7

Conclusion

This dissertation has addressed two fundamental problems for objects-first CS1 approaches. First, it has provided an answer to the question, "How does one teach an objects-first CS1?" The answer was the graphical design-centric objects-first CS1. The dissertation focused on two parts of the course; first, a model objects-first CS1 course with syllabus, and, second, a collection of classroom-tested examples for teaching an objects-first CS1.

The objects-first model CS1 course outlined in this dissertation addresses the complaint of CC2001 that all programming-first approaches focus on syntax rather than algorithms and place little weight on design, analysis, and testing. Rather, object-oriented design has become a key component of the course, thus making it design-centric.

Empirical testing was conducted on the graphical design-centric objects-first CS1 to identify the predictors of success. Contrary to past research, the analysis revealed that

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

prior programming experience was not a predictor of success. The notion that mathematics is a predictor of success for the objects-first CS1 was shown to be dubious. No effect was found for number of high school math courses. SAT math scores accounted for only 7% of the variance in course averages in the objects-first CS1. This low value is diminished considering that SAT scores in general account for approximately 23% of the variance in freshman grades. The analyses revealed no bias for gender for either success or rate of resignation. Given the problem of retention of women in CS, it is important that introductory courses do not alienate them. Similarly, there was no effect for success for either year in school or major.

Extensive multivariate analysis showed that for each measure of success, namely, course average, lab average, and exam average, measures of effort were always the strongest predictors of success. Comfort level added to the predictive value of the multivariate models and was always second to effort. Finally, academic variables such as SAT scores and critical thinking ability offered very small predictive value.

Analyses of rate of resignation were restricted, due to the fact that students who resigned generally did not participate in the questionnaires. However, the analyses revealed that students who resigned were registered for significantly more credit hours at the beginning of the semester than those who did not resign. Differences in resign rates were seen per major, namely computer science students resigned less, while business and math students resign more. In fact, math students resign most of all. However, number of credit hours registered for seemed to be the determining factor in explaining

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

resignation rates for computer science and math students. Computer science students were registered for the least number of credit hours, while math students were registered for the most. Alternate course options explained the resignation rates of the business students.

The empirical results of this dissertation make it a significant study for predictors of success for the objects-first CS1. While imperatives-first courses are well studied, no such analyses had been done for the objects-first CS1.

Overall, the results are very encouraging. Predictors of success for imperatives-first courses such as prior programming experience and mathematics do not hold for the objects-first course. The fact that students of different experiences with regard to programming do equally well is encouraging. Taken with the lack of gender bias and that all majors performing equally well, it appears that the objects-first CS1 is a CS1 for everyone. It would seem to be well-suited in a liberal arts setting as a first programming course for majors and non-majors alike.

The dissertation has included suggestions for future work. Perhaps one of the most important is for cross-institutional replication of the empirical work of the dissertation. Since this dissertation is the first, it is important that the ideas be replicated.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Chapter 8

Future Work

This chapter presents future research directions based on the findings of this dissertation.

8.1 Predictors of Success and CS2

The CS2 course at UB, CSE116, is highly dependent on the material presented in UB's CS1, CSE115. In addition to covering the typical CS2 material, such as abstract datatypes (ADTs), sorting, and searching, the course covers programming-in-the-large, additional design patterns including State-based ADTs with heavy use of Visitors (Nguyen, 1998; Nguyen & Wong, 1999), and software engineering topics such as good documentation (in terms of naming and comments). The software engineering topics have been peer-enforced through the use of code swapping. Students are given a medium-size project to work on for nearly the duration of the semester. The project is

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

broken down into three stages, with a working program due at each stage. The students work in groups. For stages 2 and 3, the students build their solution on the solution for the prior stage from a different group. Part of their grade is based on feedback about documentation and design from the group(s) using their code.

One possible extension for the present work would be to examine the effect of the various predictors of success from this dissertation for the CS2 students. One question is whether traditional factors such as prior programming (before CS1) and mathematical ability hold in the OF CS2 course.

8.2 CS1 Not Just for CS Students

At UB, the CS1 and CS2 courses are unique in that they serve primarily computer science and computer engineering majors. This was a result of a decision to provide two tracks of introductory courses, one for majors and one for non-majors. Anecdotally, we have seen many students take the non-majors versions of the course only to come to the OF CS1 and remark how suddenly it all makes sense. Several have even asked why the non-majors course was not taught the same way.

Considering the findings of this dissertation that a student's major played no role in his or her success in the OF class, as well as the fact that many departments find themselves in the situation of having only a single introductory CS class, it would be interesting to investigate how well non-majors courses could use the ideas of the graphical objects-first design-centric CS1 explored in this dissertation.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

8.3 Leading a Horse to Water and the Lost

The results of this dissertation have revealed the prime factors for success in the course to be largely a matter of a student's own personal responsibility, for instance the number of labs turned in and recitation attendance. The question naturally arises, how do we encourage students to act responsibly, i.e., attend recitation, submit all lab assignments, and take all the exams? In an attempt to encourage student attendance at recitations, undergraduate students who had recently taken CSE115 and done well were employed as teaching assistants. These undergraduate teaching assistants (UTAs) are the sole teaching staff for the recitations. This change has resulted in a huge difference in terms of the student satisfaction with their TA. Prior to the introduction of UTAs, a full two-thirds of students had significant complaints about TAs in the course. Approximately one third of the students complained of the poor English skills of the (foreign) graduate teaching assistants. The additional third not only complained about language skills but also about the fact that the graduate TAs simply did not understand the material themselves.

Currently, with the UTAs, the number of student complaints about TAs on evaluations has dropped to less than 5 in a semester! The students routinely seek out the help of the UTAs while taking CS2, which only uses graduate TAs. All of this notwithstanding, it would seem there is still room for improvement in recitation attendance.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Related to this is the phenomenon of students who simply give up trying in the course. The CS1 instructors have noticed (and in fact other instructors of freshman and sophomore level courses at UB have corroborated) large numbers of students who simply stop trying in the course, but do not resign. Attempts have been made to contact these students after the course is over to determine what might have helped them in the course. While the response rate is generally quite low, the students who do respond in general report that there was nothing that could be changed in the course, as their trouble was a result of their own acknowledged lack of work in the course. An investigation of ways of helping freshman students adapt to college life may be in order.

One additional group for which data is sparse is those students who resign the course. The question is, "Why do they resign?" Are they resigning to avoid a poor grade? If so, is it due to some lack of prerequisite knowledge or personal factors? Have they concluded that they are not really interested in the study of computer science after all? This group has been problematic to study, since they generally do not participate in the course surveys.

8.4 Multi-Institutional Analysis

The graphical objects-first design-centric CS1 presented in this dissertation is but one implementation of the CC2001 recommendations. As was noted in the background section, there are several institutions with their own take on CC2001's objects-first curriculum. While studies like the one in the current dissertation are a good first step in

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

understanding how the OF curriculum changes the nature of CS1 education, huge gains can be had through large-scale multi-institutional analysis. It would be interesting to replicate the work of this dissertation across various institutions with an objects-first curriculum. Such a study could greatly enhance the understanding of objects-first as a whole apart from any single curricular implementation.

8.5 Long-term

While the findings reported here seem to show modest benefits for the objects-first CS1, one is left to wonder what effect it will have on students' ability later in the curriculum. Will students exposed to an objects-first CS1 have difficulty in later classes? Will they perform worse than students with an imperative-first CS1? Do they lose their ability to do OOD when the rest of the curriculum is objects-neutral or even anti-OO? These questions can only be answered through a longitudinal study.

8.5.1 Preliminary Investigation

In an attempt to ascertain whether the objects-first CS1 is harmful for student performance later in the curriculum, a preliminary investigation was conducted.

8.5.1.1 Method

The subjects of the study were students from the Fall 2001 through Spring 2003 *CSE421: Operating Systems* classes at University at Buffalo, SUNY. The grades of the students for CSE421 as well as the record of their CS1 course were obtained from

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

InfoSource. For any student who took CSE421 multiple times between Fall 2001 and Spring 2003, the first grade in CSE421 was used. Only students who actually completed the course were included in the study. After filtering, there were 274 students. For each of these students, the type of CS1 course was determined. The categories were OO for objects-first CS1 taken at UB, Imp for imperative-first CS1 taken at UB, and Trans for transfer students. The largest number of students came from the imperative-first category. This is reasonable considering that the objects-first CS1 only began at UB in the Spring 2000 semester. The objects-first group was second largest, followed by the transfer students.

8.5.1.2 Results

A Kruskal-Wallis ANOVA was used on the data (see Table 8.1). As can be seen in the table, the objects-first students' rank was better than that of the other two categories. However, the difference is not statistically significant ($\chi^2(2) = .914$, $p = .63320$).

	CS 1 Type	*n*	Mean Rank
OS Grade	Imp	150	133.44
	Trans	49	132.84
	OO	72	143.49
	Total	271	

Table 8.1 OS Course Grade by CS1 Type

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

8.5.1.3 Discussion

At the very least the data support the notion that exposure to OF CS1 "does no harm." However there are a number of problems drawing any strong conclusions from the current data. First, the OF curriculum at UB started in the Spring 2000 semester, which means any OF student in the operating systems course would have gone from a first-semester freshman course to a senior-level course in at most 3 years (assuming a student took CS1 in Spring 2000 and OS in Spring 2003), and in many of the cases this time-frame is compacted. This suggests that the sample for the preliminary study may be non-representative of the OF students as a whole. Are the students in the sample over-achievers, thus inflating the results seen, or does their lack of time to mature mean that their rank is lower than will be seen with the true population? Time and future analysis will tell.

Some would consider the use of an OS course in examining the long-term effect of OF to be helpful in the sense that OS courses typically involve "low-level hardcore (imperative-style)" programming. While this may be the case in many universities and colleges, at UB the course uses Tom Anderson's *Nachos* instructional operating system (Christopher, Procter, & Anderson). Nachos is written in object-oriented C++. That is, basic OS constructs such as main memory, file systems, schedulers, etc., are represented by objects in the system. Therefore, it is still uncertain given the preliminary investigation how OF students fare in non-OO contexts.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Finally, a compelling long-term study should include an analysis of student performance across a range of courses throughout the curriculum. This will provide a more comprehensive picture of the state of affairs for OF CS1 students. Such analysis would also examine the extent to which OOP/OOD were encouraged or discouraged by the faculty member as well as student utilization of OOD techniques. Anecdotal evidence reveals that, even after a good grounding in OOP/OOD, many students will revert to imperative techniques in later courses, believing them to be quicker to implement. However, many times, these students are sadly mistaken. For instance, in the Spring 2003 semester, one of the Computer Organization instructors gave a project that involved creating a simulation of virtual memory in a machine. Students were required to implement a small language parser that allowed setup of the virtual memory simulation as well as reads and writes to/from memory. None of the students who attempted an imperative design completed the project fully, while those who used an object-oriented design either fully completed the assignment or completed a substantial amount of the work.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Bibliography

American Psychological Association. (2001). *Publication manual of the American Psychological Association* (5th ed.). Washington, DC: American Psychological Association.

Aron, A., & Aron, E. N. (2002). *Statistics for the Behavioral and Social Sciences: A Brief Course* (2nd ed.). Upper Saddle River, NJ: Prentice Hall.

Astrachan, O., Mitchener, G., Berry, G., & Cox, L. (1998). Design patterns: An essential component of CS curricula. *ACM SIGCSE Bulletin , Proceedings of the twenty-ninth SIGCSE technical symposium on Computer science education, 30*(1), 153-160.

Barnes, D. J., & Kölling, M. (2003). *Objects first with Java : a practical introduction using BlueJ*. New York: Pearson Education.

Bergin, J., Koffman, E., Proulx, V. K., Rasala, R., & Wolz, U. (1999). Objects: When, why, and how? *Journal of Computing in Small Colleges, 14*(4).

Booch, G., Rumbaugh, J., & Jacobson, I. (1999). *The Unified Modeling Language User Guide*: Addison-Wesley.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Byrne, P., & Lyons, G. (2001). The effect of student attributes on success in programming. *Proceedings of the 6th annual conference on Innovation and technology in computer science education*, 49-52.

Cantwell Wilson, B., & Shrock, S. (2001). Contributing to success in an introductory computer science course: A study of twelve factors. *ACM SIGCSE Bulletin , Proceedings of the thirty second SIGCSE technical symposium on Computer Science Education, 33*(1), 184-188.

Chotin, M. (2003). *NGP Tutorial.* Retrieved October 27, 2003, from the World Wide Web: http://www.cs.brown.edu/courses/cs015/2003/ref/ngp/

Christopher, W. A., Procter, S. J., & Anderson, T. A.*The Nachos Instructional Operating System.* Retrieved 6/21/03, 2003, from the World Wide Web: http://http.cs.berkeley.edu/~tea/nachos/nachos.ps

Culwin, F. (1999). Object Imperatives! *ACM SIGCSE Bulletin , The proceedings of the thirtieth SIGCSE technical symposium on Computer science education, 31*(1), 31-36.

Cuny, J., & Aspray, W. (2001). *Recruitment and Retention of Women Graduate Students in Computer Science and Engineering.* Washington, DC: Computing Research Association.

Curricula, T. J. T. F. o. C. (2001). *Computing Curricula 2001 Computer Science*: IEEE Computer Society & Association for Computing Machinery.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Dahl, O.-J., & Nygaard, K.*How object-oriented programming started.* Retrieved 04/06, 2002, from the World Wide Web: http://www.ifi.uio.no/~kristen/FORSKNINGSDOK_MAPPE/F_OO_start.html

Ennis, R. (1985). *Cornell Critical Thinking Tests Level X & Level Z Manual* (3rd ed.): Critical Thinking Books & Software.

Evans, G. E., & Simkin, M. G. (1989). What best predicts computer proficiency? *Communications of the ACM, 32*(11), 1322-1327.

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1995). *Design Patterns: Elements of Reusable Object-Oriented Software*: Addison-Wesley.

Gittleman, A. (2002). *Computing with Java : programs, objects, graphics* (Alternate 2nd ed.). El Granada, Calif.: Scott/Jones.

Hagan, D., & Markham, S. (2000). Does it help to have some programming experience before beginning a computing degree program? *ACM SIGCSE Bulletin , 5th annual SIGCSE/SIGCUE conference on Innovation and technology in computer science education, 32*(3), 25-28.

Holmes, B., & Joyce, D. T. (2001). *Object-oriented programming with Java* (2nd ed.). Sudbury, Mass.: Jones and Bartlett.

Howell, D. C. (1999). *Fundamental Statistics for the Behavioral Sciences* (4th ed.): Brooks/Cole.

Joy, B., Steele, G., Gosling, J., & Bracha, G. (2000). *Java Language Specification* (2nd ed.): Addison-Wesley.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Kölling, M., & Rosenberg, J. (2001). Guidelines for teaching object orientation with Java. *Proceedings of the 6th annual conference on Innovation and technology in computer science education*, 33-36.

Kurtz, B. L. (1980). Investigating the relationship between the development of abstract reasoning and performance in an introductory programming class. *The papers of the eleventh SIGCSE technical symposium on Computer science education : SIGCSE bulletin*, 110-117.

Leeper, R. R., & Silver, J. L. (1982). *Predicting success in a first programming course.* Paper presented at the Thirteenth SIGCSE Technical symposium on Computer Science Education, Indianapolis, Indiana.

Lewis, J. (2000). Myths about Object-Orientation and Its Pedagogy. *ACM SIGCSE Bulletin , Proceedings of the thirty-first SIGCSE technical symposium on Computer science education, 32*(1), 245-249.

Margolis, J., & Fisher, A. (2002). *Unlocking the Clubhouse: Women Studying Computer Science*. Cambridge, MA: MIT Press.

Mazlack, L. J. (1980). Identifying potential to acquire programming skill. *Communications of the ACM, 23*(1), 14-17.

Mitchell, W. (2001). A paradigm shift to OOP has occurred...implementation to follow. *Journal of Computing in Small Colleges, 16*(2), 95-106.

Morelli, R. (2003). *Java, Java, Java! : object-oriented problem solving* (2nd ed.). Upper Saddle River, NJ: Prentice Hall.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Nguyen, D. Z. (1998). Design patterns for data structures. *ACM SIGCSE Bulletin , Proceedings of the twenty-ninth SIGCSE technical symposium on Computer science education, 30*(1), 336-340.

Nguyen, D. Z. (2002). *Comp 212 Spring 2002 Lecture 1.* Retrieved October 11, 2003, from the World Wide Web: http://www.owlnet.rice.edu/~comp212/03-spring/lectures/01/

Nguyen, D. Z., & Wong, S. B. (1999). Patterns for decoupling data structures and algorithms. *ACM SIGCSE Bulletin , The proceedings of the thirtieth SIGCSE technical symposium on Computer science education, 31*(1), 87-91.

Nguyen, D. Z., & Wong, S. B. (2001). *OOP in introductory CS: Better students through abstraction.* Paper presented at the Fifth Workshop on and Tools for Assimilating Object-Oriented Concepts, OOPSLA01, Tampa, Florida.

Nino, J., & Hosch, F. A. (2002). *An introduction to programming and object-oriented design using JAVA.* New York: Wiley.

Norusis, M. J. (2002). *SPSS 11.0 Guide to Data Analysis.* Upper Saddle River, NJ: Prentice Hall.

Nowaczyk, R. H. (1983). *Cognitive skills needed in computer programming.* Paper presented at the Annual Meeting of the Southeastern Psychological Association, Atlanta, Georgia.

Perry, J. E. (1996). *An introduction to object-oriented design in C++*: Addison-Wesley.

Ramalingam, V., & Wiedenbeck, S. (1998). Development and validation of scores on a computer programming self-efficacy scale and group analyses of novice

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

programmer self-efficacy. *Journal of Educational Computing Research, 19*(4), 367-381.

Riley, D. D. (2002). *The object of Java : introduction to programming using software engineering principles*. Boston: Addison Wesley.

Roberts, E. S., Kassianidou, M., & Irani, L. (2002). Encouraging Women in Computer Science. *SIGCSE Bulletin, 34*(2), 84-88.

Sheard, J., & Hagan, D. (1998). Our failing students: A study of a repeat group. *ACM SIGCSE Bulletin , Proceedings of the 6th annual conference on the teaching of computing/3rd annual conference on integrating technology into computer science education on Changing the delivery of computer science education, 30*(3), 223-227.

Steele Hanson, A. M. (1986). *Critical Thinking Ability of Novice and Expert Computer Programmers*. University of Idaho.

van Dam, A. (2002). *CS015 Home Page* [Web page]. Retrieved September 2, 2002, from the World Wide Web: http://www.cs.brown.edu/courses/cs015/

van Dam, A. (2003). *CS015 Advanced NGP Lecture Notes* [Web]. Retrieved October 12, 2003, from the World Wide Web: http://www.cs.brown.edu/courses/cs015/2003/lecture/foils/09_AdvancedNGP4.pdf

Walingford, E. (1996). Toward a first course based on object-oriented patterns. *ACM SIGCSE Bulletin , Proceedings of the twenty-seventh SIGCSE technical symposium on Computer science education, 28*(1).

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Weber-Wulff, D. (2000). Combating the code warrior: A different sort of programming instruction. *ACM SIGCSE Bulletin , 5th annual SIGCSE/SIGCUE conference on Innovation and technology in computer science education, 32*(3), 85-88.

Weisert, C. (2002). Pseudo Object-Oriented Programming Considered Harmful. *ACM SIGPLAN Notices, 37*(4), 31.

Westfall, R. (2001). Hello, Worl Considered Harmful. *Communications of the ACM, 44*(10), 129-130.

Wong, S. B.*CS150 Lecture 1 Notes* [Web page]. Retrieved 9/18, 2003, from the World Wide Web: http://www.exciton.cs.rice.edu/cs150/notes/lec01.htm

Woodman, M., Davis, G., & Holland, S. (1996). The joy of software -- starting with objects. *ACM SIGCSE Bulletin , Proceedings of the twenty-seventh SIGCSE technical symposium on Computer science education, 28*(1), 88-92.

Wu, C. T. (2001). *An introduction to object-oriented programming with Java* (2nd ed.). Boston, Mass.: McGraw Hill.

Xing, C.-c., & Belkhouche, B. (2003). On Pseudo Object-Oriented Programming Considered Harmful. *Communications of the ACM, 46*(10), 115-117.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Appendix A

Meyers-Briggs Major Pairs

Four Major Pairs of the Myers-Briggs Traits (Evans & Simkin, 1989)

Extrovert vs Introvert	
Extroversion probably means you relate more easily to the outer world of people and things than to the inner world of ideas. You like variety and action; are often good at greeting people; are often impatient with long slow jobs; often act quickly, sometimes without thinking; like to have people around; and usually communicate freely.	*Introversion* means you relate more easily to the inner world of ideas than to the outer world of people and things. You like quiet for concentration; tend to be careful with details; dislike sweeping statements; have trouble remembering names and faces; dislike telephone intrusions and interruptions; work contentedly alone; and have some problems communicating.
Sensing vs Intuitive	
Sensing means you would rather work with known facts than look for new possibilities and relationships. You dislike new problems unless there are standard ways to solve them; like an established way of doing things; enjoy using skills already learned more than learning new ones; seldom make errors of fact; tend to be good at precise work; and are patient with routine details.	*Intuitive* means you would rather look for possibilities and relationships than work with known facts. You like solving new problems; dislike doing the same thing repeatedly; enjoy learning a new skill more than using it; work in bursts of energy powered by enthusiasm, with slack periods in between; reach a conclusion quickly; and are impatient with routine details.
Thinking vs Feeling	

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Thinking means you base your judgments more on impersonal analysis and logic than on personal values. You do not show emotion readily and are often uncomfortable dealing with people's feelings without knowing it; like analysis and putting things into logical order; tend to decide impersonally, sometimes paying insufficient attention to people's wishes; and are able to reprimand people or fire them when necessary.

Feeling means you base your judgments more on personal values than on impersonal analysis and logic. You tend to be very aware of other people and their feelings; enjoy pleasing people, even in unimportant things; dislike telling people unpleasant things; tend to be sympathetic; and like harmony.

Judging vs Perceiving

Judging means you like a planned, decided, orderly way of life better than a flexible, spontaneous way. You work best when you can plan your work and follow the plans; like to get things settled and finished; may decide things too quickly; and may dislike to interrupt the project you are on for a more urgent one.

Perceiving means you like a flexible, spontaneous way of life better than a planned, decided, orderly way. You adapt well to changing situations; do not mind leaving things open for alterations; may have trouble making decisions; and may start too many projects and have difficulty in finishing them.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Appendix B

Correlation Matrix

Key
*   Significant at .05
**   Significant at .01

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

APPENDIX B CORRELATION MATRIX

	Credit Hours	Num Hours per Week at Job	Log of Number of Hours Worked at Job	HS Avg
Credit Hours	1	-0.413**	-0.297**	0.217**
Num Hours per Week at Job	-0.413**	1	0.913**	-0.042
Log of Number of Hours	-0.297**	0.913**	1	-0.047
HS Avg	0.217**	-0.042	-0.047	1
HS Percentile Rank	0.197**	-0.139	-0.113	0.749**
Years HS Math	0.208**	-0.096	-0.083	0.216**
SAT Math	0.172**	-0.138*	-0.165*	0.491**
SAT Verbal	0.061	-0.126*	-0.144*	0.392**
Office Hour Visits	-0.070	-0.039	-0.027	0.054
Log of Number of Office	-0.062	-0.035	-0.010	0.056
Percent Recitation Time	-0.011	0.013	-0.003	0.219**
Log of Percent Recitation	0.100*	-0.057	-0.031	0.206**
Comfort Level	0.033	-0.021	0.020	0.257**
Cornell CT Score	0.074	0.005	-0.014	0.108
SE - Independence	0.080	0.000	0.000	0.069
SE - Complex Programming	0.083	0.016	0.031	0.076
SE - Self-Regulation	0.061	0.008	0.012	0.257**
SE - Simple Programming	0.095	-0.017	-0.013	0.087
Att - Exam Difficulty	0.130*	-0.094	-0.070	0.111
Att - Luck	0.046	0.064	0.080	-0.132
Att - Effort	0.037	0.060	0.036	0.104
Att - Ability	0.038	-0.006	0.039	0.145*

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

APPENDIX B CORRELATION MATRIX

	HS Percentile Rank	Years HS Math	SAT Math	SAT Verbal	Hour Visits
Credit Hours	0.197**	0.208**	0.172**	0.061	-0.070
Num Hours per Week at Job	-0.139	-0.096	-0.138*	-0.126*	-0.039
Log of Number of Hours	-0.113	-0.083	-0.165*	-0.144*	-0.027
HS Avg	0.749**	0.216**	0.491**	0.392**	0.054
HS Percentile Rank	1	0.156*	0.314**	0.206**	0.108
Years HS Math	0.156*	1	0.249**	0.092	-0.017
SAT Math	0.314**	0.249**	1	0.539**	-0.131**
SAT Verbal	0.206**	0.092	0.539**	1	-0.092*
Office Hour Visits	0.108	-0.017	-0.131**	-0.092*	1
Log of Number of Office	0.133*	-0.068	-0.122*	-0.091*	0.898**
Percent Recitation Time	0.218**	-0.042	0.003	-0.085	0.405**
Log of Percent Recitation	0.201**	0.019	0.040	-0.010	0.325**
Comfort Level	0.170*	-0.006	0.178*	0.103	0.142*
Cornell CT Score	0.133	0.144*	0.390**	0.457**	-0.033
SE - Independence	-0.095	0.074	0.204**	0.043	-0.210**
SE - Complex Programming	-0.078	0.050	0.317**	0.108	-0.234**
SE - Self-Regulation	0.073	0.052	0.223**	0.025	-0.191**
SE - Simple Programming	-0.068	0.113*	0.307**	0.174*	-0.195**
Att - Exam Difficulty	0.212*	0.079	-0.055	-0.062	0.068
Att - Luck	-0.090	-0.004	-0.003	0.035	-0.135*
Att - Effort	0.040	0.036	-0.009	-0.007	0.055
Att - Ability	0.030	0.006	0.034	-0.022	-0.035

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

APPENDIX B CORRELATION MATRIX

	Log of Number of Office Hour Visits	Recitation Time Usage	Log of Percent Recitation Time Usage	Comfort Level
Credit Hours	-0.062	-0.011	0.100*	0.033
Num Hours per Week at Job	-0.035	0.013	-0.057	-0.021
Log of Number of Hours	-0.010	-0.003	-0.031	0.020
HS Avg	0.056	0.219**	0.206**	0.257**
HS Percentile Rank	0.133*	0.218**	0.201**	0.170*
Years HS Math	-0.068	-0.042	0.019	-0.006
SAT Math	-0.122*	0.003	0.040	0.178*
SAT Verbal	-0.091*	-0.085	-0.010	0.103
Office Hour Visits	0.898**	0.405**	0.325**	0.142*
Log of Number of Office	1	0.478**	0.417**	0.215**
Percent Recitation Time	0.478**	1	0.759**	0.177**
Log of Percent Recitation	0.417**	0.759**	1	0.202**
Comfort Level	0.215**	0.177**	0.202**	1
Cornell CT Score	-0.039	0.009	0.033	0.177**
SE - Independence	-0.265**	-0.131*	-0.105*	0.176**
SE - Complex Programming	-0.288**	-0.151*	-0.120*	0.242**
SE - Self-Regulation	-0.176**	-0.013	0.034	0.324**
SE - Simple Programming	-0.224**	-0.128*	-0.060	0.259**
Att - Exam Difficulty	0.043	0.134*	0.117*	-0.119*
Att - Luck	-0.137*	-0.069	-0.046	-0.196**
Att - Effort	0.076	0.092	0.081	0.005
Att - Ability	-0.039	-0.074	-0.007	-0.002

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

APPENDIX B CORRELATION MATRIX

	CT Score	SE - Independence Persistence	SE - Complex Programming	SE – Self-Regulation
Credit Hours	0.074	0.080	0.083	0.061
Num Hours per Week at Job	0.005	0.000	0.016	0.008
Log of Number of Hours	-0.014	0.000	0.031	0.012
HS Avg	0.108	0.069	0.076	0.257**
HS Percentile Rank	0.133	-0.095	-0.078	0.073
Years HS Math	0.144*	0.074	0.050	0.052
SAT Math	0.390**	0.204**	0.317**	0.223**
SAT Verbal	0.457**	0.043	0.108	0.025
Office Hour Visits	-0.033	-0.210**	-0.234**	-0.191**
Log of Number of Office	-0.039	-0.265**	-0.288**	-0.176**
Percent Recitation Time	0.009	-0.131*	-0.151**	-0.013
Log of Percent Recitation	0.033	-0.105*	-0.120**	0.034
Comfort Level	0.177**	0.176**	0.242**	0.324**
Cornell CT Score	1	0.070	0.088	0.038
SE - Independence	0.070	1	0.882**	0.765**
SE - Complex Programming	0.088	0.882**	1	0.750**
SE - Self-Regulation	0.038	0.765**	0.750**	1
SE - Simple Programming	0.175**	0.814**	0.812**	0.700**
Att - Exam Difficulty	0.052	-0.105*	-0.149**	-0.047
Att - Luck	-0.083	0.105*	0.113*	-0.015
Att - Effort	0.106	-0.010	-0.043	0.025
Att - Ability	-0.019	0.051	0.013	0.114*

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

APPENDIX B CORRELATION MATRIX

	SE – Simple Programming	Att - Exam Difficulty	Att - Luck	Att - Effort	Att - Ability	C (in years)
Credit Hours	0.095	0.130*	0.046	0.037	0.038	0.069
Num Hours per Week at Job	-0.017	-0.094	0.064	0.060	-0.006	0.003
Log of Number of Hours	-0.013	-0.070	0.080	0.036	0.039	-0.004
HS Avg	0.087	0.111	-0.132	0.104	0.145	-0.105
HS Percentile Rank	-0.068	0.212*	-0.090	0.040	0.030	-0.259**
Years HS Math	0.113*	0.079	-0.004	0.036	0.006	0.065
SAT Math	0.307**	-0.055	-0.003	-0.009	0.034	0.150*
SAT Verbal	0.174*	-0.062	0.035	-0.007	-0.022	0.179**
Office Hour Visits	-0.195**	0.068	-0.135*	0.055	-0.035	-0.124*
Log of Number of Office	-0.224**	0.043	-0.137*	0.076	-0.039	-0.158**
Percent Recitation Time	-0.128*	0.134*	-0.069	0.092	-0.074	-0.086
Log of Percent Recitation	-0.060	0.117*	-0.046	0.081	-0.007	-0.100
Comfort Level	0.259**	-0.119*	-0.196**	0.005	-0.002	0.126*
Cornell CT Score	0.175**	0.052	-0.083	0.106	-0.019	0.043
SE - Independence	0.814**	-0.105*	0.105*	-0.010	0.051	0.275**
SE - Complex Programming	0.812**	-0.149**	0.113*	-0.043	0.013	0.346**
SE - Self-Regulation	0.700**	-0.047	-0.015	0.025	0.114*	0.191**
SE - Simple Programming	1	-0.148**	0.014	0.056	0.034	0.298**
Att - Exam Difficulty	-0.148**	1	0.001	0.082	0.300**	-0.140*
Att - Luck	0.014	0.001	1	-0.184**	-0.082	0.139*
Att - Effort	0.056	0.082	-0.184**	1	0.180**	-0.128*
Att - Ability	0.034	0.300**	-0.082	0.180**	1	-0.173**

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

	Log of C (in years)	C++ (in years)	Prior C++?	Log of C++ (in years)	Java (in years)	Prior Java?
Credit Hours	0.080	0.119*	0.162**	0.138*	-0.047	-0.066
Num Hours per Week at Job	0.012	-0.084	-0.096	-0.094	0.003	0.064
Log of Number of Hours	0.023	-0.074	-0.064	-0.078	-0.029	0.041
HS Avg	-0.083	0.022	0.107	0.066	-0.094	-0.079
HS Percentile Rank	-0.229**	-0.093	0.006	-0.055	-0.105	-0.010
Years HS Math	0.073	0.011	0.042	0.018	-0.057	-0.049
SAT Math	0.166*	0.282**	0.351**	0.323**	-0.060	-0.122
SAT Verbal	0.186**	0.201**	0.173*	0.208**	-0.103	-0.173*
Office Hour Visits	-0.144**	-0.171**	-0.187**	-0.182**	-0.003	0.025
Log of Number of Office	-0.169**	-0.219**	-0.211**	-0.222**	-0.023	0.041
Percent Recitation Time	-0.098	-0.151**	-0.104*	-0.136*	-0.071	-0.068
Log of Percent Recitation	-0.114*	-0.120*	-0.065	-0.093	-0.092	-0.094
Comfort Level	0.169**	0.094	0.133*	0.120*	0.093	0.075
Cornell CT Score	0.036	0.083	0.049	0.089	-0.142*	-0.150*
SE - Independence	0.297**	0.512**	0.516**	0.536**	0.190**	0.097
SE - Complex Programming	0.371**	0.580**	0.537**	0.596**	0.199**	0.080
SE - Self-Regulation	0.230**	0.388**	0.466**	0.436**	0.159**	0.135*
SE - Simple Programming	0.325**	0.551**	0.568**	0.583**	0.142*	0.062
Att - Exam Difficulty	-0.114*	-0.134*	-0.089	-0.119*	-0.080	-0.016
Att - Luck	0.127*	0.047	0.038	0.041	0.226**	0.149**
Att - Effort	-0.129*	-0.014	0.003	-0.003	-0.113*	-0.044
Att - Ability	-0.149**	-0.031	0.021	-0.005	-0.079	-0.016

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

APPENDIX B CORRELATION MATRIX

	Log of Java (in years)	Pascal (in years)	Log of Pascal (in years)	Basic (in years)	Log of Basic (in years)
Credit Hours	-0.062	0.076	0.097	0.061	0.029
Num Hours per Week at Job	0.025	0.110*	0.068	0.010	0.010
Log of Number of Hours	-0.004	0.090	0.055	-0.021	-0.018
HS Avg	-0.090	-0.077	-0.042	-0.038	-0.020
HS Percentile Rank	-0.082	-0.072	-0.066	-0.194*	-0.175*
Years HS Math	-0.055	0.002	0.008	0.083	0.115*
SAT Math	-0.084	0.091	0.128*	0.294**	0.329**
SAT Verbal	-0.131	0.096	0.102	0.246**	0.259**
Office Hour Visits	0.004	-0.036	-0.059	-0.127*	-0.142*
Log of Number of Office	-0.009	-0.005	-0.029	-0.173**	-0.197**
Percent Recitation Time	-0.075	-0.035	-0.013	-0.171**	-0.177**
Log of Percent Recitation	-0.103	-0.013	0.001	-0.142*	-0.152**
Comfort Level	0.088	0.091	0.091	0.182**	0.187**
Cornell CT Score	-0.153	0.083	0.100	0.214**	0.239**
SE - Independence	0.176	0.216**	0.264**	0.368**	0.435**
SE - Complex Programming	0.176	0.238**	0.292**	0.420**	0.506**
SE - Self-Regulation	0.158	0.203**	0.226**	0.244**	0.301**
SE - Simple Programming	0.124	0.215**	0.265**	0.388**	0.459**
Att - Exam Difficulty	-0.056	-0.087	-0.035	-0.164**	-0.137*
Att - Luck	0.213	-0.071	-0.073	-0.022	-0.002
Att - Effort	-0.095	-0.019	-0.044	-0.021	-0.040
Att - Ability	-0.053	-0.099	-0.081	-0.044	0.009

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

APPENDIX B CORRELATION MATRIX

	HTML (in years)	Log of HTML (in years)	Scripting (in years)	Log of Scripting (in years)	Ttl Yrs Prog
Credit Hours	-0.043	-0.014	0.010	-0.025	0.032
Num Hours per Week at Job	0.108*	0.089	0.056	0.082	0.054
Log of Number of Hours	0.066	0.062	0.050	0.079	0.017
HS Avg	-0.017	-0.003	-0.069	-0.053	-0.080
HS Percentile Rank	-0.100	-0.069	-0.120	-0.099	-0.235**
Years HS Math	0.018	-0.010	0.017	-0.004	0.066
SAT Math	0.130*	0.122	0.162*	0.132*	0.292**
SAT Verbal	0.235**	0.237**	0.191**	0.177**	0.269**
Office Hour Visits	-0.159**	-0.192**	-0.133*	-0.139*	-0.170**
Log of Number of Office	-0.187**	-0.207**	-0.160**	-0.161**	-0.220**
Percent Recitation Time	-0.210**	-0.229**	-0.158**	-0.162**	-0.208**
Log of Percent Recitation	-0.276**	-0.285**	-0.177**	-0.177**	-0.204**
Comfort Level	0.094	0.130*	0.025	0.028	0.173**
Cornell CT Score	0.189**	0.171**	0.090	0.076	0.184**
SE - Independence	0.272**	0.306**	0.211**	0.225**	0.485**
SE - Complex Programming	0.291**	0.335**	0.234**	0.275**	0.546**
SE - Self-Regulation	0.178**	0.222**	0.058	0.089	0.332**
SE - Simple Programming	0.349**	0.370**	0.244**	0.265**	0.523**
Att - Exam Difficulty	-0.135*	-0.125*	-0.252**	-0.259**	-0.258**
Att - Luck	0.045	0.117*	0.034	0.056	0.027
Att - Effort	-0.070	-0.117*	-0.085	-0.122*	-0.073
Att - Ability	-0.020	-0.056	-0.143*	-0.143*	-0.114*

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

APPENDIX B CORRELATION MATRIX

	Log of Total Year Programming	Programming Experience?	Num Prog Lang	Programming Languages	Percent Labs Submitted
Credit Hours	0.035	-0.040	0.034	0.006	0.121**
Num Hours per Week at Job	0.030	0.039	0.030	0.035	-0.036
Log of Number of Hours	0.016	0.049	0.024	0.032	-0.011
HS Avg	0.029	0.108	-0.011	0.050	0.301**
HS Percentile Rank	-0.134	0.054	-0.127	-0.047	0.218**
Years HS Math	0.098	0.152**	0.088	0.113*	0.131*
SAT Math	0.341**	0.170*	0.311**	0.313**	0.177**
SAT Verbal	0.264**	0.044	0.189**	0.165*	0.012
Office Hour Visits	-0.182**	-0.023	-0.156**	-0.134*	0.287**
Log of Number of Office	-0.239**	-0.046	-0.165**	-0.143**	0.344**
Percent Recitation Time	-0.209**	-0.067	-0.158**	-0.135*	0.492**
Log of Percent Recitation	-0.197**	-0.060	-0.173**	-0.148**	0.595**
Comfort Level	0.239**	0.197**	0.244**	0.262**	0.211**
Cornell CT Score	0.182**	0.073	0.115*	0.111	0.043
SE - Independence	0.625**	0.497**	0.582**	0.620**	0.006
SE - Complex Programming	0.684**	0.490**	0.650**	0.665**	-0.003
SE - Self-Regulation	0.495**	0.506**	0.502**	0.569**	0.106*
SE - Simple Programming	0.700**	0.597**	0.646**	0.700**	0.011
Att - Exam Difficulty	-0.176**	0.021	-0.161**	-0.111*	-0.041
Att - Luck	0.031	0.063	0.103	0.090	-0.086
Att - Effort	-0.080	0.004	-0.100	-0.073	0.064
Att - Ability	-0.032	0.026	-0.062	-0.023	0.023

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

APPENDIX B CORRELATION MATRIX

	Exams Taken	Lab Avg	Exam Avg	Course Avg
Credit Hours	0.141**	0.110*	0.117*	0.130**
Num Hours per Week at Job	-0.029	-0.048	-0.050	-0.054
Log of Number of Hours	0.021	-0.031	-0.062	-0.053
HS Avg	0.286**	0.308**	0.391**	0.389**
HS Percentile Rank	0.283**	0.241**	0.327**	0.314**
Years HS Math	0.070	0.073	0.052	0.074
SAT Math	0.114*	0.241**	0.334**	0.324**
SAT Verbal	0.040	0.102	0.214**	0.191**
Office Hour Visits	0.198**	0.351**	0.220**	0.291**
Log of Number of Office	0.261**	0.411**	0.293**	0.358**
Percent Recitation Time	0.382**	0.540**	0.438**	0.524**
Log of Percent Recitation	0.546**	0.583**	0.519**	0.600**
Comfort Level	0.062	0.453**	0.418**	0.464**
Cornell CT Score	0.016	0.136*	0.169**	0.190**
SE - Independence	0.103	-0.012	0.072	0.063
SE - Complex Programming	0.045	-0.004	0.084	0.076
SE - Self-Regulation	0.127*	0.140*	0.189**	0.200**
SE - Simple Programming	0.064	0.080	0.164**	0.155**
Att - Exam Difficulty	0.108*	-0.085	-0.030	-0.054
Att - Luck	0.027	-0.184**	-0.194**	-0.189**
Att - Effort	-0.034	0.031	0.046	0.057
Att - Ability	0.095	0.028	0.101	0.076

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

APPENDIX B CORRELATION MATRIX

	Credit Hours	Week at Job	Worked at Job	HS Avg	Percentil e Rank	Years HS Math
C (in years)	0.069	0.003	-0.004	-0.105	-0.259**	0.065
Log of C (in years)	0.080	0.012	0.023	-0.083	-0.229**	0.073
C++ (in years)	0.119*	-0.084	-0.074	0.022	-0.093	0.011
Prior C++?	0.162**	-0.096	-0.064	0.107	0.006	0.042
Log of C++ (in years)	0.138*	-0.094	-0.078	0.066	-0.055	0.018
Java (in years)	-0.047	0.003	-0.029	-0.094	-0.105	-0.057
Prior Java?	-0.066	0.064	0.041	-0.079	-0.010	-0.049
Log of Java (in years)	-0.062	0.025	-0.004	-0.090	-0.082	-0.055
Pascal (in years)	0.076	0.110	0.090	-0.077	-0.072	0.002
Log of Pascal (in years)	0.097	0.068	0.055	-0.042	-0.066	0.008
Basic (in years)	0.061	0.010	-0.021	-0.038	-0.194*	0.083
Log of Basic (in years)	0.029	0.010	-0.018	-0.020	-0.175*	0.115*
HTML (in years)	-0.043	0.108*	0.066	-0.017	-0.100	0.018
Log of HTML (in years)	-0.014	0.089	0.062	-0.003	-0.069	-0.010
Scripting (in years)	0.010	0.056	0.050	-0.069	-0.120	0.017
Log of Scripting (in years)	-0.025	0.082	0.079	-0.053	-0.099	-0.004

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

APPENDIX B CORRELATION MATRIX

	SAT Math	SAT Verbal	Office Hour Visits	Log of Number of Office Hour Visits
C (in years)	0.150*	0.179**	-0.124*	-0.158**
Log of C (in years)	0.166*	0.186**	-0.144**	-0.169**
C++ (in years)	0.282**	0.201**	-0.171**	-0.219**
Prior C++?	0.351**	0.173*	-0.187**	-0.211**
Log of C++ (in years)	0.323**	0.208**	-0.182**	-0.222**
Java (in years)	-0.060	-0.103	-0.003	-0.023
Prior Java?	-0.122	-0.173*	0.025	0.041
Log of Java (in years)	-0.084	-0.131*	0.004	-0.009
Pascal (in years)	0.091	0.096	-0.036	-0.005
Log of Pascal (in years)	0.128*	0.102	-0.059	-0.029
Basic (in years)	0.294**	0.246**	-0.127*	-0.173**
Log of Basic (in years)	0.329**	0.259**	-0.142*	-0.197**
HTML (in years)	0.130*	0.235**	-0.159**	-0.187**
Log of HTML (in years)	0.122	0.237**	-0.192**	-0.207**
Scripting (in years)	0.162*	0.191**	-0.133*	-0.160**
Log of Scripting (in years)	0.132*	0.177**	-0.139*	-0.161**

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

APPENDIX B CORRELATION MATRIX

	Percent Recitation Time Usage	Log of Percent Recitation Time Usage	Comfort Level
C (in years)	-0.086	-0.100	0.126*
Log of C (in years)	-0.098	-0.114*	0.169**
C++ (in years)	-0.151**	-0.120*	0.094
Prior C++?	-0.104*	-0.065	0.133*
Log of C++ (in years)	-0.136*	-0.093	0.120*
Java (in years)	-0.071	-0.092	0.093
Prior Java?	-0.068	-0.094	0.075
Log of Java (in years)	-0.075	-0.103*	0.088
Pascal (in years)	-0.035	-0.013	0.091
Log of Pascal (in years)	-0.013	0.001	0.091
Basic (in years)	-0.171**	-0.142**	0.182**
Log of Basic (in years)	-0.177**	-0.152**	0.187**
HTML (in years)	-0.210**	-0.276**	0.094
Log of HTML (in years)	-0.229**	-0.285**	0.130*
Scripting (in years)	-0.158**	-0.177**	0.025
Log of Scripting (in years)	-0.162**	-0.177**	0.028

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

APPENDIX B CORRELATION MATRIX

	Cornell CT Score	Independence Persistence	SE - Complex Programming	SE - Self-Regulation
C (in years)	0.043	0.275**	0.346**	0.191**
Log of C (in years)	0.036	0.297**	0.371**	0.230**
C++ (in years)	0.083	0.512**	0.580**	0.388**
Prior C++?	0.049	0.516**	0.537**	0.466**
Log of C++ (in years)	0.089	0.536**	0.596**	0.436**
Java (in years)	-0.142*	0.190**	0.199**	0.159**
Prior Java?	-0.150*	0.097	0.080	0.135*
Log of Java (in years)	-0.153*	0.176**	0.176**	0.158**
Pascal (in years)	0.083	0.216**	0.238**	0.203**
Log of Pascal (in years)	0.100	0.264**	0.292**	0.226**
Basic (in years)	0.214**	0.368**	0.420**	0.244**
Log of Basic (in years)	0.239**	0.435**	0.506**	0.301**
HTML (in years)	0.189**	0.272**	0.291**	0.178**
Log of HTML (in years)	0.171**	0.306**	0.335**	0.222**
Scripting (in years)	0.090	0.211**	0.234**	0.058
Log of Scripting (in years)	0.076	0.225**	0.275**	0.089

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

APPENDIX B CORRELATION MATRIX

	SE - Simple Programming	Att - Exam Difficulty	Att - Luck	Att - Effort	Att - Ability
C (in years)	0.298**	-0.140*	0.139*	-0.128*	-0.173**
Log of C (in years)	0.325**	-0.114*	0.127*	-0.129*	-0.149**
C++ (in years)	0.551**	-0.134*	0.047	-0.014	-0.031
Prior C++?	0.568**	-0.089	0.038	0.003	0.021
Log of C++ (in years)	0.583**	-0.119*	0.041	-0.003	-0.005
Java (in years)	0.142*	-0.080	0.226**	-0.113*	-0.079
Prior Java?	0.062	-0.016	0.149**	-0.044	-0.016
Log of Java (in years)	0.124*	-0.056	0.213**	-0.095	-0.053
Pascal (in years)	0.215**	-0.087	-0.071	-0.019	-0.099
Log of Pascal (in years)	0.265**	-0.035	-0.073	-0.044	-0.081
Basic (in years)	0.388**	-0.164**	-0.022	-0.021	-0.044
Log of Basic (in years)	0.459**	-0.137*	-0.002	-0.040	0.009
HTML (in years)	0.349**	-0.135*	0.045	-0.070	-0.020
Log of HTML (in years)	0.370**	-0.125*	0.117*	-0.117*	-0.056
Scripting (in years)	0.244**	-0.252**	0.034	-0.085	-0.143*
Log of Scripting (in years)	0.265**	-0.259**	0.056	-0.122*	-0.143*

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

APPENDIX B CORRELATION MATRIX

	C (in years)	Log of C (in years)	C++ (in years)	Prior C++?	Log of C++ (in years)
C (in years)	1	0.900**	0.236**	0.131*	0.204**
Log of C (in years)	0.900**	1	0.314**	0.246**	0.301**
C++ (in years)	0.236**	0.314**	1	0.794**	0.978**
Prior C++?	0.131*	0.246**	0.794**	1	0.891**
Log of C++ (in years)	0.204**	0.301**	0.978**	0.891**	1
Java (in years)	0.354**	0.226**	0.039	-0.047	0.003
Prior Java?	0.084	0.028	-0.099	-0.110*	-0.121*
Log of Java (in years)	0.262**	0.164**	0.002	-0.065	-0.031
Pascal (in years)	0.140*	0.158**	0.132*	0.023	0.095
Log of Pascal (in years)	0.205**	0.230**	0.232**	0.094	0.190**
Basic (in years)	0.431**	0.373**	0.276**	0.171**	0.252**
Log of Basic (in years)	0.358**	0.357**	0.369**	0.256**	0.350**
HTML (in years)	0.172**	0.221**	0.218**	0.101	0.185**
Log of HTML (in years)	0.182**	0.243**	0.257**	0.131*	0.225**
Scripting (in years)	0.339**	0.291**	0.185**	0.042	0.143*
Log of Scripting (in years)	0.348**	0.310**	0.197**	0.039	0.148**

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

APPENDIX B CORRELATION MATRIX

	Java (in years)	Prior Java?	Log of Java (in years)	Pascal (in years)	Log of Pascal (in years)
C (in years)	0.354**	0.084	0.262	0.140*	0.205**
Log of C (in years)	0.226**	0.028	0.164	0.158**	0.230**
C++ (in years)	0.039	-0.099	0.002	0.132*	0.232**
Prior C++?	-0.047	-0.110*	-0.065	0.023	0.094
Log of C++ (in years)	0.003	-0.121*	-0.031	0.095	0.190**
Java (in years)	1	0.824**	0.984**	-0.061	-0.052
Prior Java?	0.824**	1	0.901**	-0.091	-0.097
Log of Java (in years)	0.984**	0.901**	1	-0.071	-0.066
Pascal (in years)	-0.061	-0.091	-0.071	1	0.910**
Log of Pascal (in years)	-0.052	-0.097	-0.066	0.910**	1
Basic (in years)	0.221**	0.020	0.160**	0.434**	0.355**
Log of Basic (in years)	0.149**	-0.022	0.101	0.292**	0.286**
HTML (in years)	0.074	-0.010	0.062	0.202**	0.260**
Log of HTML (in years)	0.077	-0.008	0.066	0.197**	0.236**
Scripting (in years)	0.249**	0.091	0.208**	0.106*	0.131*
Log of Scripting (in years)	0.265**	0.098	0.222**	0.157**	0.173**

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

	Basic (in years)	Log of Basic (in years)	HTML (in years)	Log of HTML (in years)
C (in years)	0.431**	0.358**	0.172**	0.182**
Log of C (in years)	0.373**	0.357**	0.221**	0.243**
C++ (in years)	0.276**	0.369**	0.218**	0.257**
Prior C++?	0.171**	0.256**	0.101	0.131*
Log of C++ (in years)	0.252**	0.350**	0.185**	0.225**
Java (in years)	0.221**	0.149**	0.074	0.077
Prior Java?	0.020	-0.022	-0.010	-0.008
Log of Java (in years)	0.160**	0.101	0.062	0.066
Pascal (in years)	0.434**	0.292**	0.202**	0.197**
Log of Pascal (in years)	0.355**	0.286**	0.260**	0.236**
Basic (in years)	1	0.913**	0.394**	0.371**
Log of Basic (in years)	0.913**	1	0.433**	0.418**
HTML (in years)	0.394**	0.433**	1	0.945**
Log of HTML (in years)	0.371**	0.418**	0.945**	1
Scripting (in years)	0.394**	0.363**	0.541**	0.510**
Log of Scripting (in years)	0.397**	0.356**	0.572**	0.558**

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

APPENDIX B CORRELATION MATRIX

	Scripting (in years)	Log of Scripting (in years)	Ttl Yrs Prog	Log of Total Year Programming
C (in years)	0.339**	0.348**	0.634**	0.504**
Log of C (in years)	0.291**	0.310**	0.580**	0.560**
C++ (in years)	0.185**	0.197**	0.451**	**0.608**
Prior C++?	0.042	0.039	0.272**	0.491**
Log of C++ (in years)	0.143*	0.148**	0.408**	0.602**
Java (in years)	0.249**	0.265**	0.329**	0.237**
Prior Java?	0.091	0.098	0.076	0.045
Log of Java (in years)	0.208**	0.222**	0.256**	0.191**
Pascal (in years)	0.106*	0.157**	0.495**	0.384**
Log of Pascal (in years)	0.131*	0.173**	0.492**	0.439**
Basic (in years)	0.394**	0.397**	0.868**	0.717**
Log of Basic (in years)	0.363**	0.356**	0.793**	0.783**
HTML (in years)	0.541**	0.572**	0.540**	0.548**
Log of HTML (in years)	0.510**	0.558**	0.495**	0.545**
Scripting (in years)	1	0.959**	0.614**	0.502**
Log of Scripting (in years)	0.959**	1	0.627**	0.530**

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

APPENDIX B CORRELATION MATRIX

	Prior Programming Experience?	Num Prog Lang	Log of Number of Programming Languages
C (in years)	0.174**	0.462**	0.388**
Log of C (in years)	0.239**	0.574**	0.498**
C++ (in years)	0.344**	0.555**	0.539**
Prior C++?	0.434**	0.553**	0.585**
Log of C++ (in years)	0.386**	0.574**	0.575**
Java (in years)	0.209**	0.351**	0.306**
Prior Java?	0.253**	0.249**	0.251**
Log of Java (in years)	0.228**	0.333**	0.300**
Pascal (in years)	0.123*	0.305**	0.270**
Log of Pascal (in years)	0.171**	0.404**	0.356**
Basic (in years)	0.234**	0.547**	0.470**
Log of Basic (in years)	0.330**	0.635**	0.580**
HTML (in years)	0.235**	0.457**	0.403**
Log of HTML (in years)	0.265**	0.466**	0.429**
Scripting (in years)	0.160**	0.448**	0.358**
Log of Scripting (in years)	0.188**	0.493**	0.401**

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

	Percent Labs Submitted	Num Exams Taken	Lab Avg	Exam Avg	Course Avg
C (in years)	-0.073	-0.021	-0.033	0.055	0.004
Log of C (in years)	-0.081	-0.002	-0.025	0.080	0.026
C++ (in years)	-0.007	-0.024	-0.010	0.043	0.032
Prior C++?	0.040	-0.002	0.056	0.074	0.071
Log of C++ (in years)	0.026	-0.003	0.023	0.074	0.066
Java (in years)	0.055	-0.046	0.023	-0.053	-0.044
Prior Java?	0.019	-0.110*	-0.002	-0.110*	-0.102
Log of Java (in years)	0.047	-0.061	0.009	-0.078	-0.067
Pascal (in years)	-0.027	0.022	0.007	0.023	0.007
Log of Pascal (in years)	-0.049	0.022	0.021	0.050	0.030
Basic (in years)	0.038	-0.055	0.041	0.054	0.054
Log of Basic (in years)	0.041	-0.029	0.032	0.061	0.062
HTML (in years)	-0.042	-0.054	0.014	0.013	0.024
Log of HTML (in years)	-0.049	-0.031	-0.008	0.002	0.010
Scripting (in years)	-0.128*	-0.131*	-0.047	-0.054	-0.049
Log of Scripting (in years)	-0.116*	-0.111*	-0.042	-0.049	-0.044

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

	Credit Hours	Week at Job	Log of Number of Hours Worked at Job	HS Avg
Ttl Yrs Prog	0.032	0.054	0.017	-0.080
Log of Total Year Programming	0.035	0.030	0.016	0.029
Prior Programming Experience?	-0.040	0.039	0.049	0.108
Num Prog Lang	0.034	0.030	0.024	-0.011
Log of Number of Programming Languages	0.006	0.035	0.032	0.050
Percent Labs Submitted	0.121**	-0.036	-0.011	0.301**
Num Exams Taken	0.141**	-0.029	0.021	0.286**
Lab Avg	0.110*	-0.048	-0.031	0.308**
Exam Avg	0.117*	-0.050	-0.062	0.391**
Course Avg	0.130**	-0.054	-0.053	0.389**

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

APPENDIX B CORRELATION MATRIX

	HS Percentile Rank	Years HS Math	SAT Math	SAT Verbal	Office Hour Visits
Ttl Yrs Prog	-0.235**	0.066	0.292**	0.269**	-0.170**
Log of Total Year Programming	-0.134	0.098	0.341**	0.264**	-0.182**
Prior Programming Experience?	0.054	0.152**	0.170*	0.044	-0.023
Num Prog Lang	-0.127	0.088	0.311**	0.189**	-0.156**
Log of Number of Programming Languages	-0.047	0.113**	0.313**	0.165*	-0.134*
Percent Labs Submitted	0.218**	0.131**	0.177**	0.012	0.287**
Num Exams Taken	0.283**	0.070*	0.114*	0.040	0.198**
Lab Avg	0.241**	0.073**	0.241**	0.102	0.351**
Exam Avg	0.327**	0.052**	0.334**	0.214**	0.220**
Course Avg	0.314**	0.074**	0.324**	0.191**	0.291**

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

	Log of Number of Office Hour Visits	Percent Recitation Time Usage	Log of Percent Recitation Time Usage
Ttl Yrs Prog	-0.220**	-0.208**	-0.204**
Log of Total Year Programming	-0.239**	-0.209**	-0.197**
Prior Programming Experience?	-0.046	-0.067	-0.060
Num Prog Lang	-0.165**	-0.158**	-0.173**
Log of Number of Programming Languages	-0.143**	-0.135*	-0.148**
Percent Labs Submitted	0.344**	0.492**	0.595**
Num Exams Taken	0.261**	0.382**	0.546**
Lab Avg	0.411**	0.540**	0.583**
Exam Avg	0.293**	0.438**	0.519**
Course Avg	0.358**	0.524**	0.600**

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

APPENDIX B CORRELATION MATRIX

	Comfort Level	Cornell CT Score	SE - Independence Persistence	SE - Complex Programming
Ttl Yrs Prog	0.173**	0.184**	0.485**	0.546**
Log of Total Year Programming	0.239**	0.182**	0.625**	0.684**
Prior Programming Experience?	0.197**	0.073	0.497**	0.490**
Num Prog Lang	0.244**	0.115*	0.582**	0.650**
Log of Number of Programming Languages	0.262**	0.111	0.620**	0.665**
Percent Labs Submitted	0.211**	0.043	0.006	-0.003
Num Exams Taken	0.062	0.016	0.103	0.045
Lab Avg	0.453**	0.136*	-0.012	-0.004
Exam Avg	0.418**	0.169**	0.072	0.084
Course Avg	0.464**	0.190**	0.063	0.076

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

APPENDIX B CORRELATION MATRIX

	SE - Self-Regulation	SE - Simple Programming	Att - Exam Difficulty	Att - Luck
Ttl Yrs Prog	0.332**	0.523**	-0.258**	0.027
Log of Total Year Programming	0.495**	0.700**	-0.176**	0.031
Prior Programming Experience?	0.506**	0.597**	0.021	0.063
Num Prog Lang	0.502**	0.646**	-0.161**	0.103
Log of Number of Programming Languages	0.569**	0.700**	-0.111*	0.090
Percent Labs Submitted	0.106*	0.011	-0.041	-0.086
Num Exams Taken	0.127*	0.064	0.108*	0.027
Lab Avg	0.140*	0.080	-0.085	-0.184**
Exam Avg	0.189**	0.164**	-0.030	-0.194**
Course Avg	0.200**	0.155**	-0.054	-0.189**

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

APPENDIX B CORRELATION MATRIX

	Att - Effort	Att - Ability	C (in years)	Log of C (in years)	C++ (in years)	Prior C++?
Ttl Yrs Prog	-0.073	-0.114*	0.634**	0.580**	0.451**	0.272**
Log of Total Year Programming	-0.080	-0.032	0.504**	0.560**	0.608**	0.491**
Prior Programming Experience?	0.004	0.026	0.174**	0.239**	0.344**	0.434**
Num Prog Lang	-0.100	-0.062	0.462**	0.574**	0.555**	0.553**
Log of Number of Programming Languages	-0.073	-0.023	0.388**	0.498**	0.539**	0.585**
Percent Labs Submitted	0.064	0.023	-0.073	-0.081	-0.007	0.040
Num Exams Taken	-0.034	0.095	-0.021	-0.002	-0.024	-0.002
Lab Avg	0.031	0.028	-0.033	-0.025	-0.010	0.056
Exam Avg	0.046	0.101	0.055	0.080	0.043	0.074
Course Avg	0.057	0.076	0.004	0.026	0.032	0.071

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

	Log of C++ (in years)	Java (in years)	Prior Java?	Log of Java (in years)
Ttl Yrs Prog	0.408**	0.329**	0.076	0.256**
Log of Total Year Programming	0.602**	0.237**	0.045	0.191**
Prior Programming Experience?	0.386**	0.209**	0.253**	0.228**
Num Prog Lang	0.574**	0.351**	0.249**	0.333**
Log of Number of Programming Languages	0.575**	0.306**	0.251**	0.300**
Percent Labs Submitted	0.026	0.055	0.019	0.047
Num Exams Taken	-0.003	-0.046	-0.110*	-0.061
Lab Avg	0.023	0.023	-0.002	0.009
Exam Avg	0.074	-0.053	-0.110*	-0.078
Course Avg	0.066	-0.044	-0.102	-0.067

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

APPENDIX B CORRELATION MATRIX

	Pascal (in years)	Log of Pascal (in years)	Basic (in years)	Log of Basic (in years)
Ttl Yrs Prog	0.495**	0.492**	0.868**	0.793**
Log of Total Year Programming	0.384**	0.439**	0.717**	0.783**
Prior Programming Experience?	0.123*	0.171**	0.234**	0.330**
Num Prog Lang	0.305**	0.404**	0.547**	0.635**
Log of Number of Programming Languages	0.270**	0.356**	0.470**	0.580**
Percent Labs Submitted	-0.027	-0.049	0.038	0.041
Num Exams Taken	0.022	0.022	-0.055	-0.029
Lab Avg	0.007	0.021	0.041	0.032
Exam Avg	0.023	0.050	0.054	0.061
Course Avg	0.007	0.030	0.054	0.062

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

APPENDIX B CORRELATION MATRIX

	HTML (in years)	Log of HTML (in years)	Scripting (in years)	Log of Scripting (in years)
Ttl Yrs Prog	0.540**	0.495**	0.614**	0.627**
Log of Total Year Programming	0.548**	0.545**	0.502**	0.530**
Prior Programming Experience?	0.235**	0.265**	0.160**	0.188**
Num Prog Lang	0.457**	0.466**	0.448**	0.493**
Log of Number of Programming Languages	0.403**	0.429**	0.358**	0.401**
Percent Labs Submitted	-0.042	-0.049	-0.128*	-0.116*
Num Exams Taken	-0.054	-0.031	-0.131*	-0.111*
Lab Avg	0.014	-0.008	-0.047	-0.042
Exam Avg	0.013	0.002	-0.054	-0.049
Course Avg	0.024	0.010	-0.049	-0.044

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

APPENDIX B CORRELATION MATRIX

	Ttl Yrs Prog	Log of Total Year Programming	Prior Programming Experience?	Num Prog Lang
Ttl Yrs Prog	1	0.863**	0.324**	0.732**
Log of Total Year Programming	0.863**	1	0.616**	0.859**
Prior Programming Experience?	0.324**	0.616**	1	0.605**
Num Prog Lang	0.732**	0.859**	0.605**	1
Log of Number of Programming Languages	0.637**	0.857**	0.793**	0.955**
Percent Labs Submitted	-0.027	-0.014	0.026	0.011
Num Exams Taken	-0.059	0.019	0.034	-0.009
Lab Avg	0.017	0.012	0.056	0.072
Exam Avg	0.059	0.108*	0.077	0.094
Course Avg	0.040	0.074	0.058	0.074

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

	Log of Number of Programming Languages	Percent Labs Submitted	Num Exams Taken
Ttl Yrs Prog	0.637**	-0.027	-0.059
Log of Total Year Programming	0.857**	-0.014	0.019
Prior Programming Experience?	0.793**	0.026	0.034
Num Prog Lang	0.955**	0.011	-0.009
Log of Number of Programming Languages	1	0.0186	0.0295
Percent Labs Submitted	0.019	1	0.699**
Num Exams Taken	0.030	0.699**	1
Lab Avg	0.069	0.879**	0.597**
Exam Avg	0.114*	0.718**	0.798**
Course Avg	0.086	0.855**	0.776**

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

APPENDIX B CORRELATION MATRIX

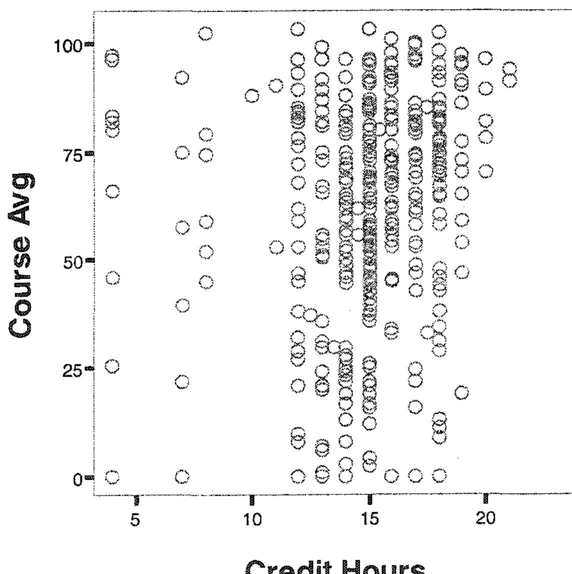
	Lab Avg	Exam Avg	Course Avg
Ttl Yrs Prog	0.017	0.059	0.040
Log of Total Year Programming	0.012	0.108*	0.074
Prior Programming Experience?	0.056	0.077	0.058
Num Prog Lang	0.072	0.094	0.074
Log of Number of Programming Languages	0.0692	0.114	0.0864
Percent Labs Submitted	0.879**	0.718**	0.855**
Num Exams Taken	0.597**	0.798**	0.776**
Lab Avg	1	0.781**	0.919**
Exam Avg	0.781**	1	0.943**
Course Avg	0.919**	0.943**	1

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

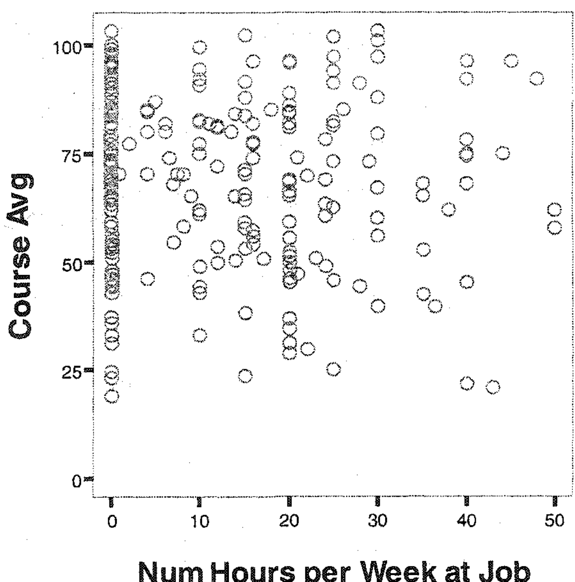
Appendix C

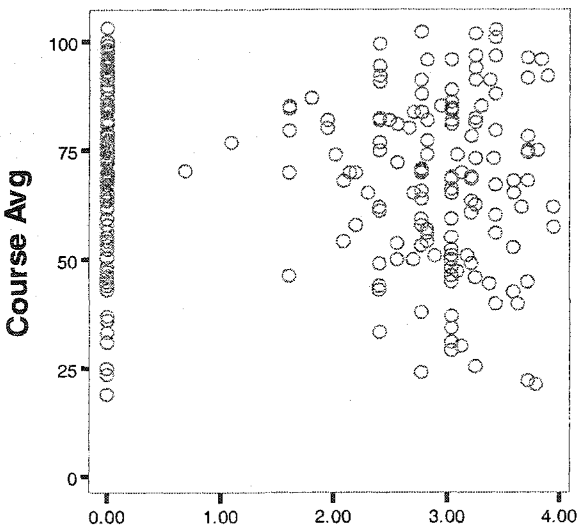
Course Average Scatterplots

Course Avg by Credit Hours

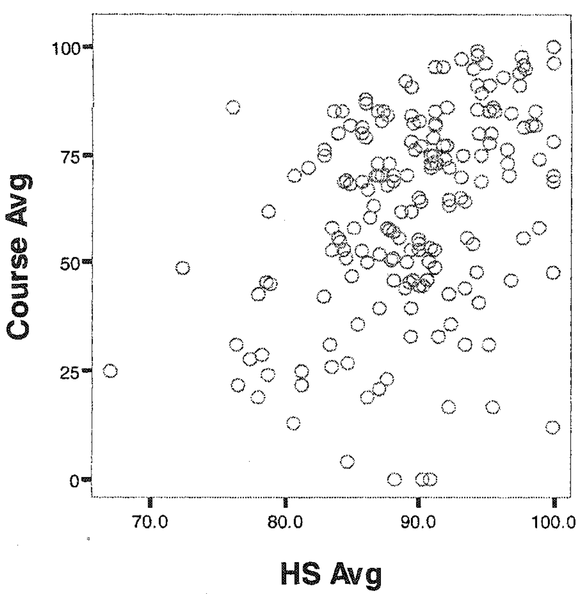


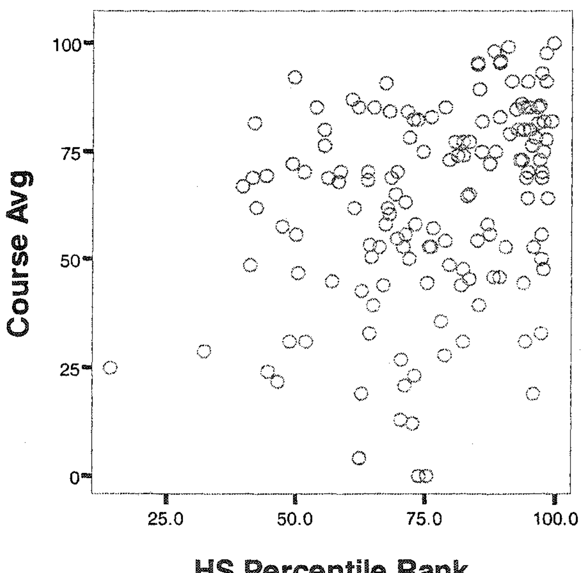
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.



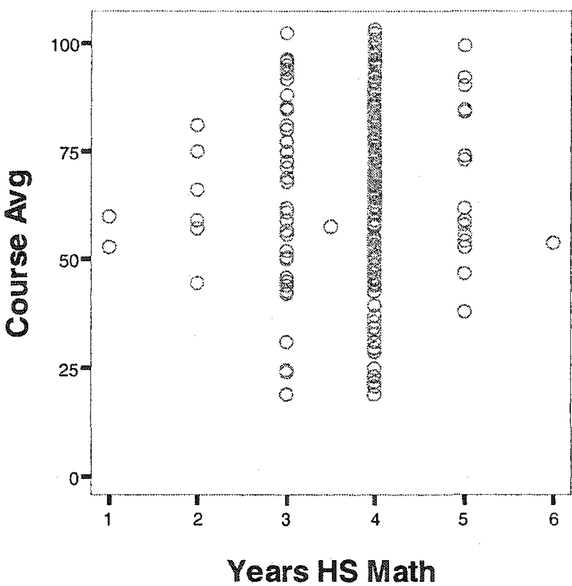


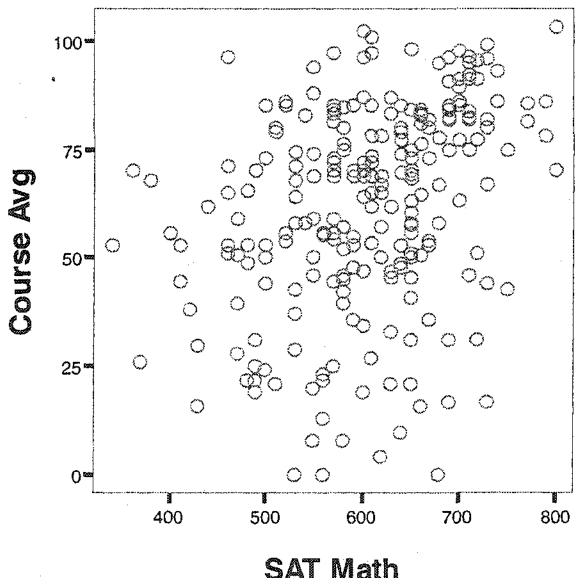
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.



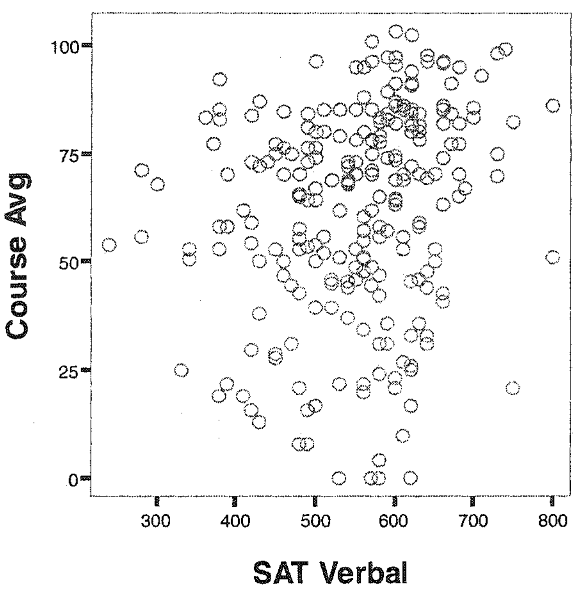


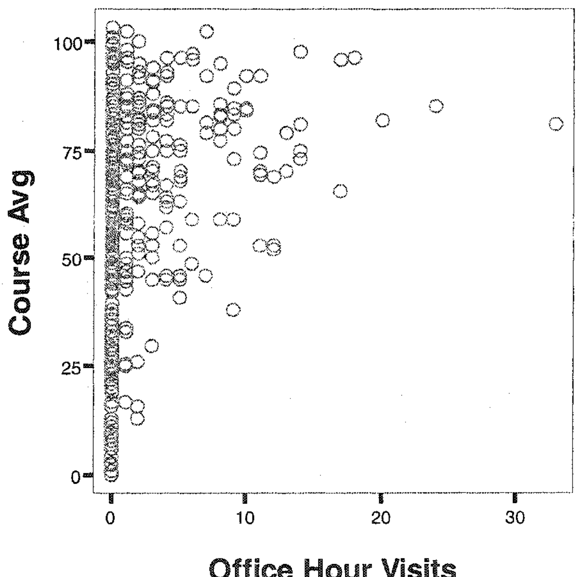
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.





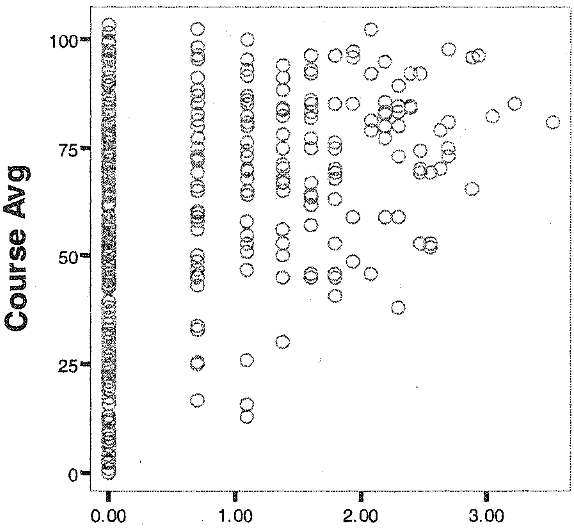
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.



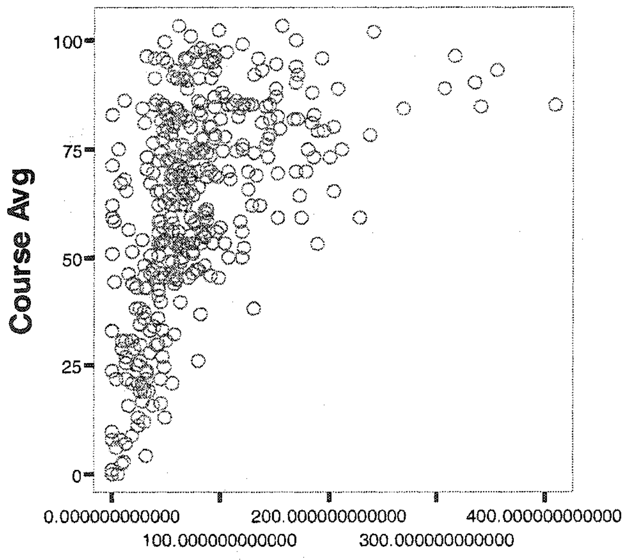


Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

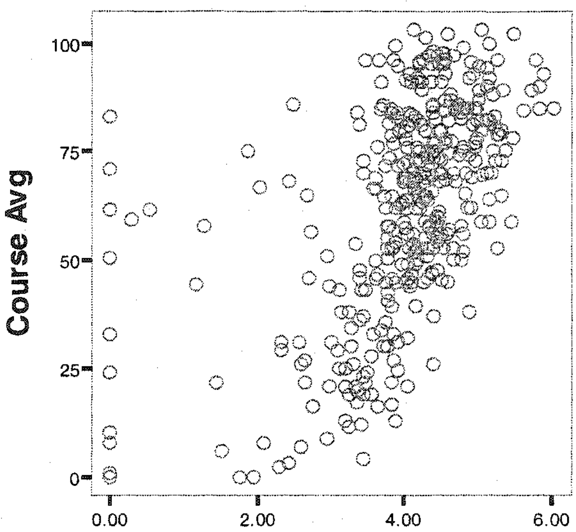
Course Avg by Log of Number of Office Hour Visits

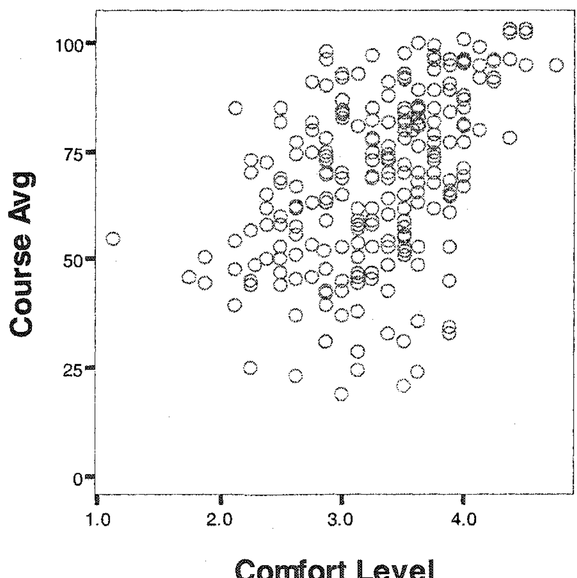


Course Avg by Percent Recitation Time Usage



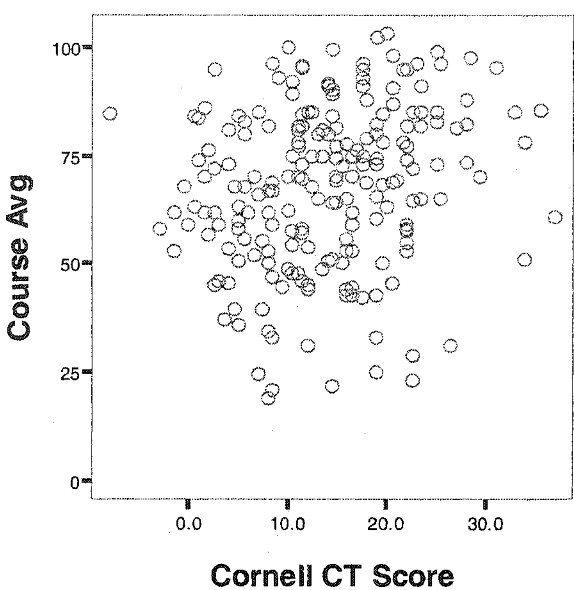
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.



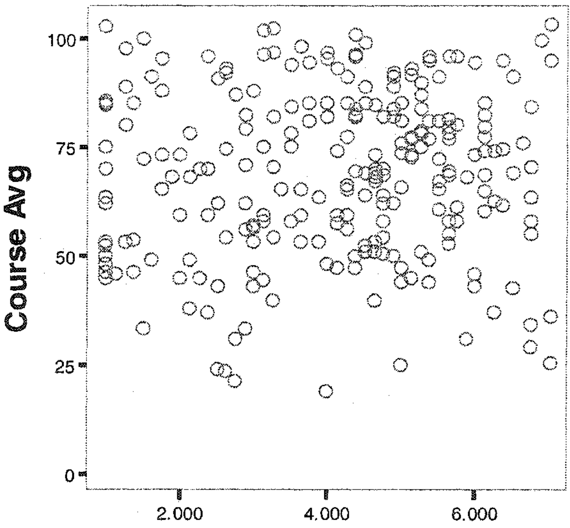


Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Course Avg by Cornell CT Score

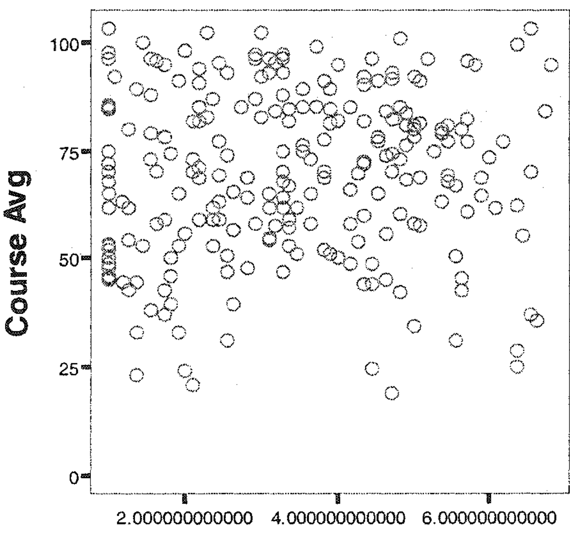


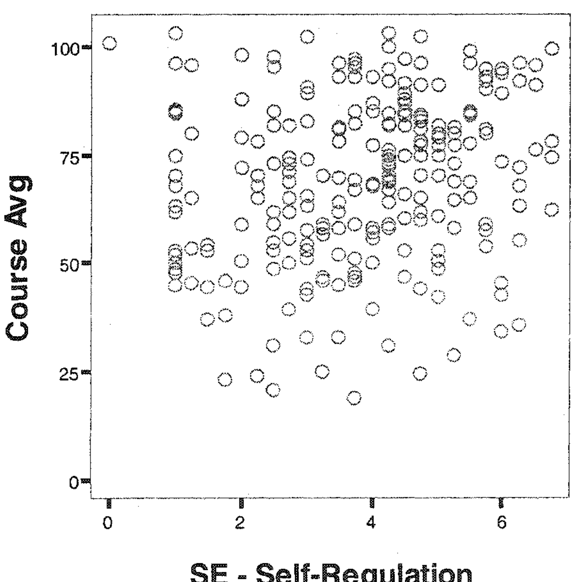
Course Avg by SE - Independence Persistence



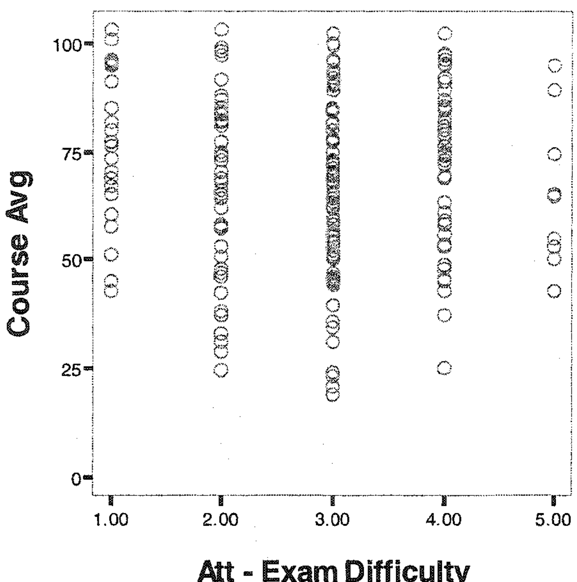
SE - Independence Persistence

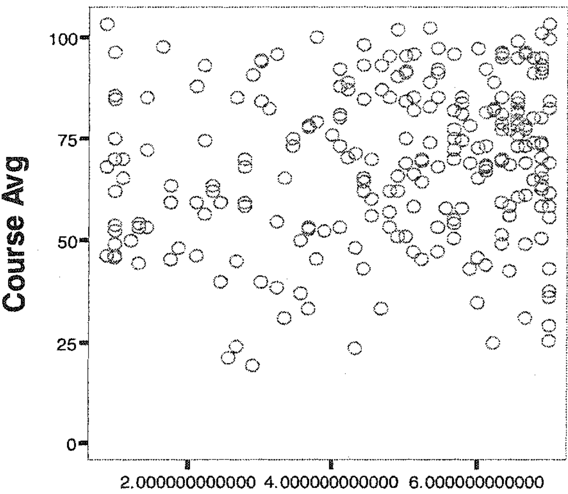
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.



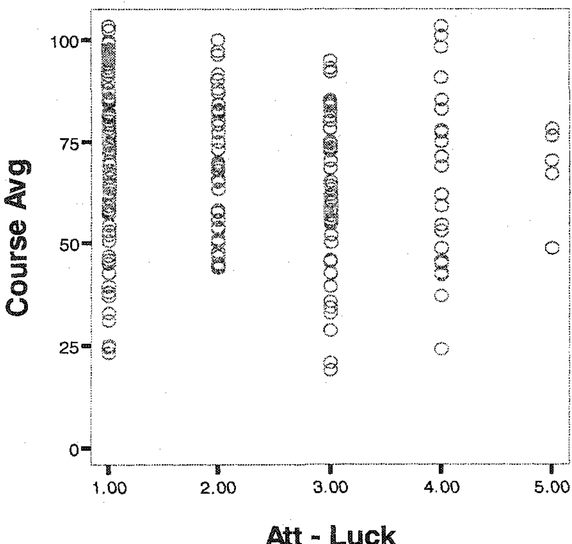


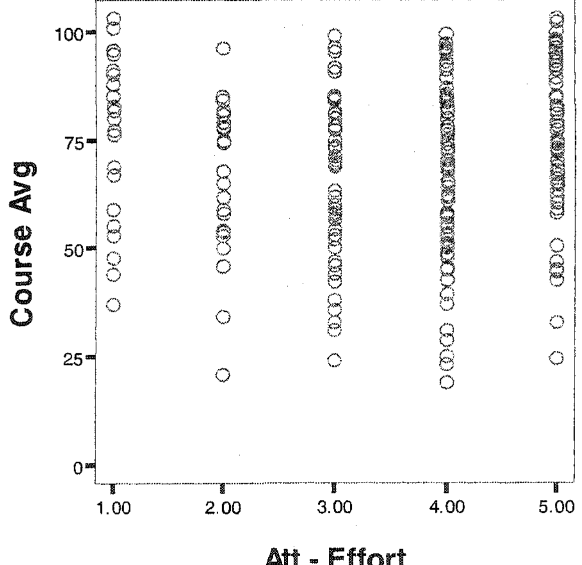
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.



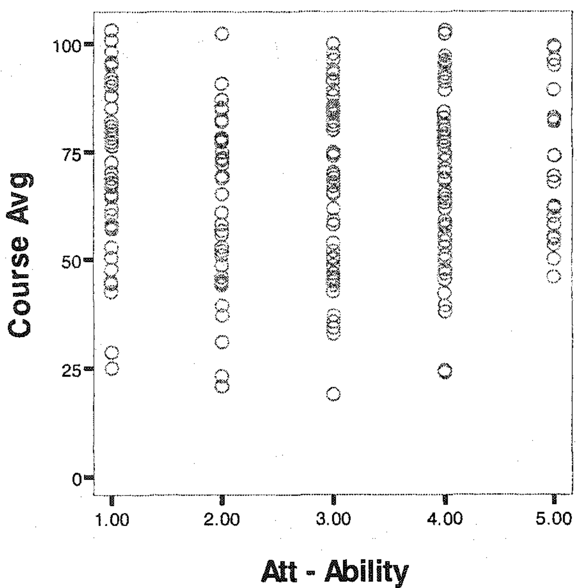


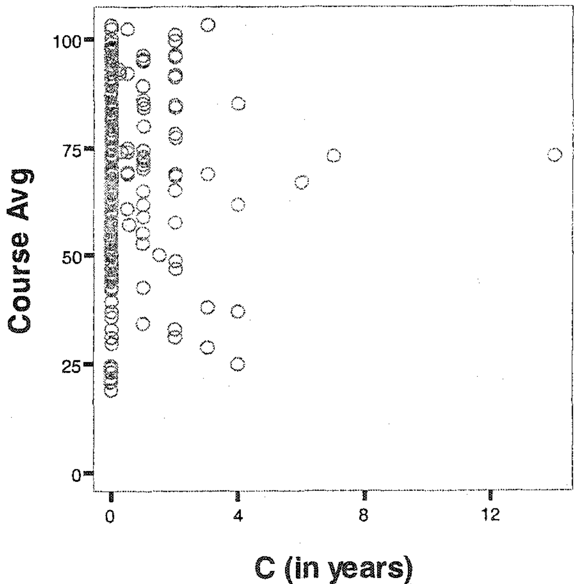
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.



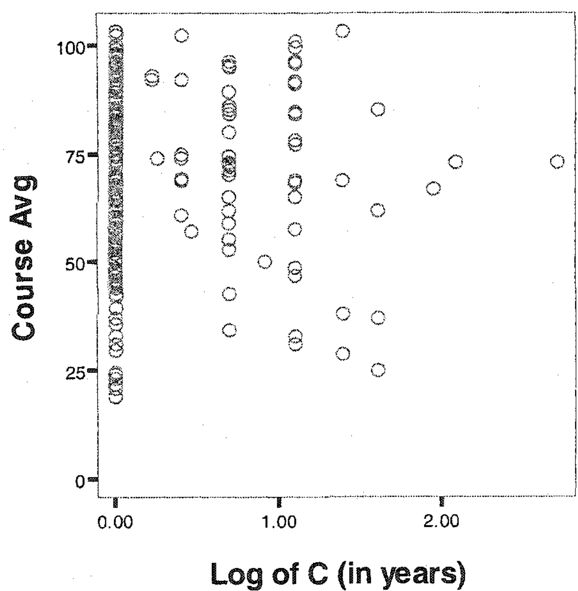


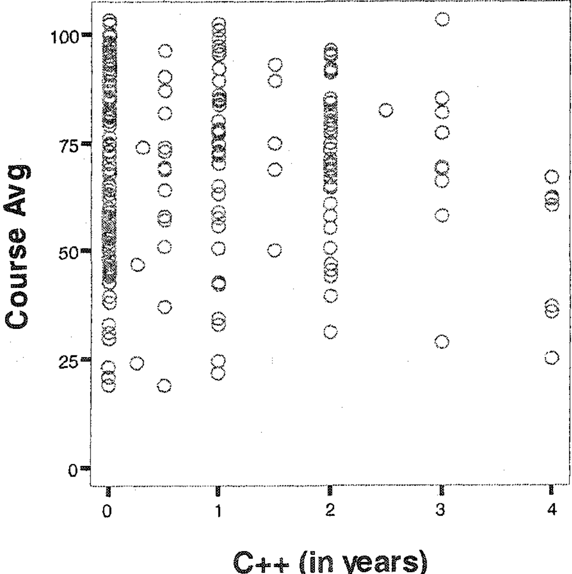
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.



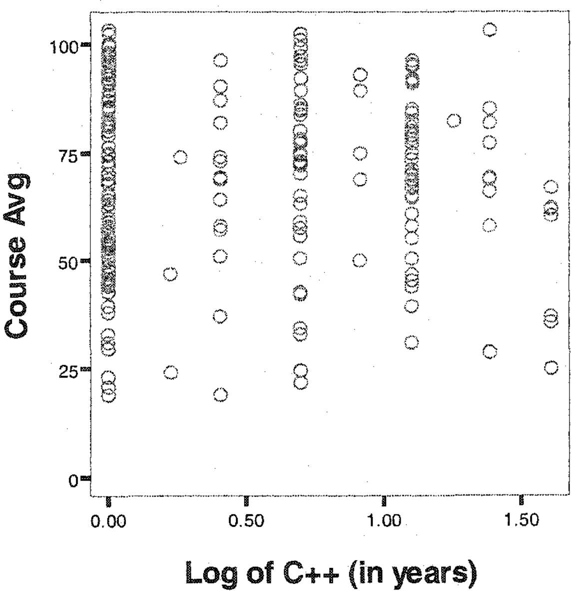


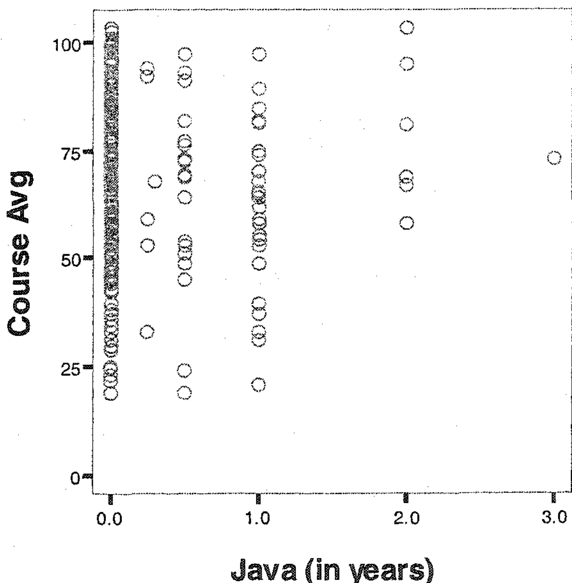
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.



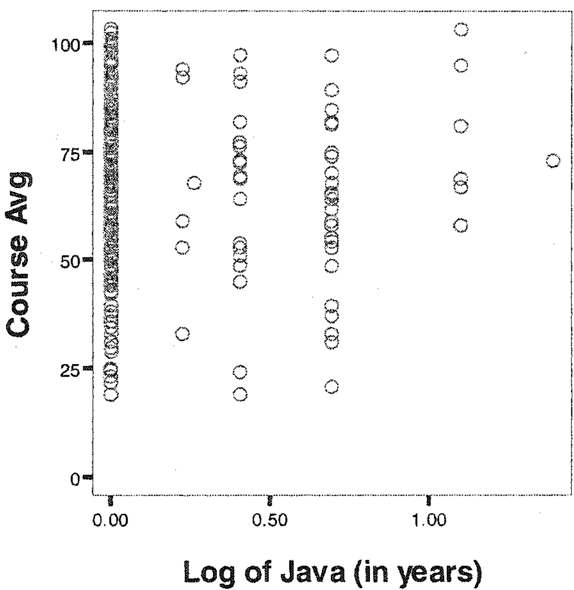


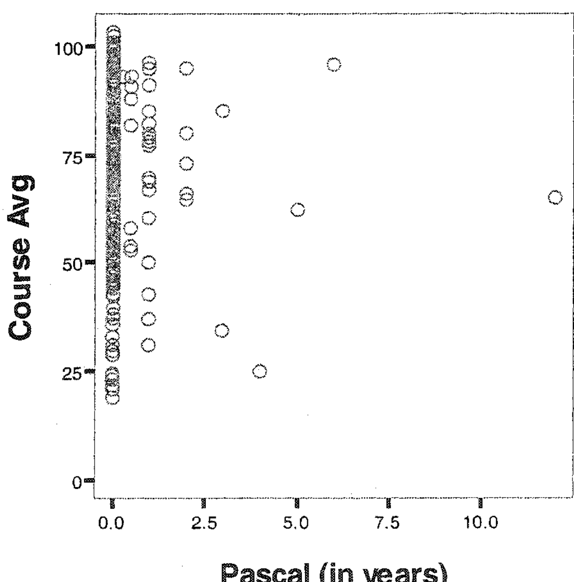
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.





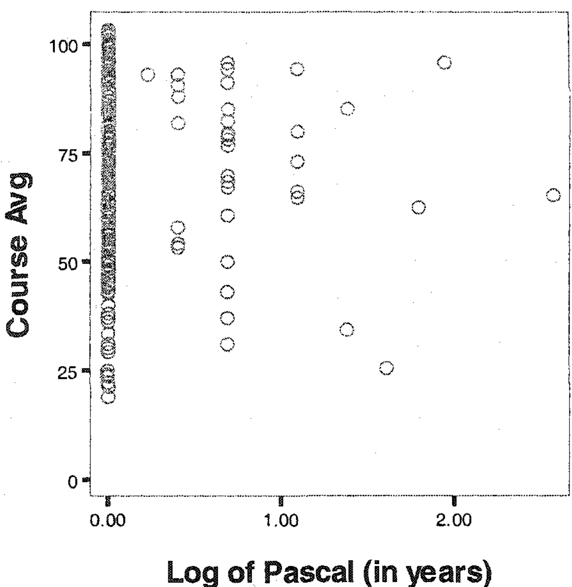
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.



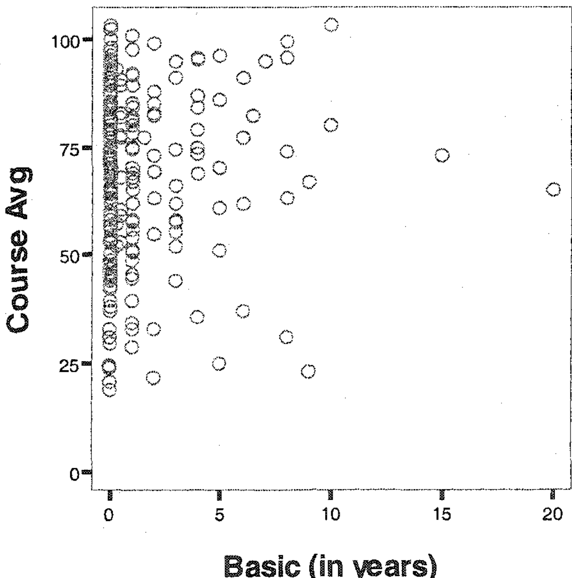


Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

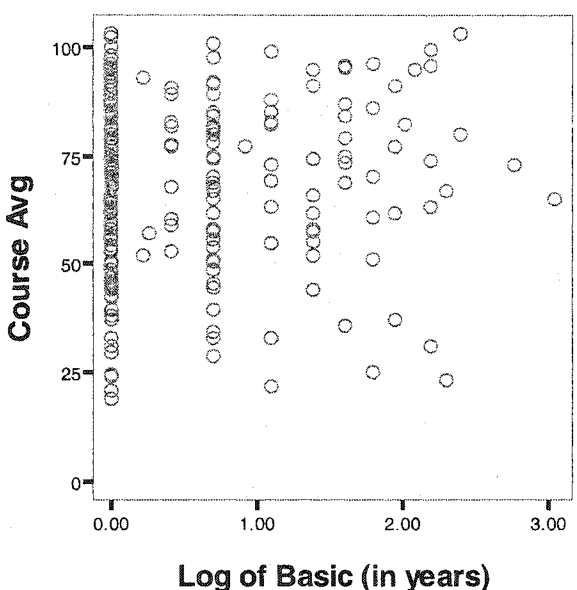
Course Avg by Log of Pascal (in years)

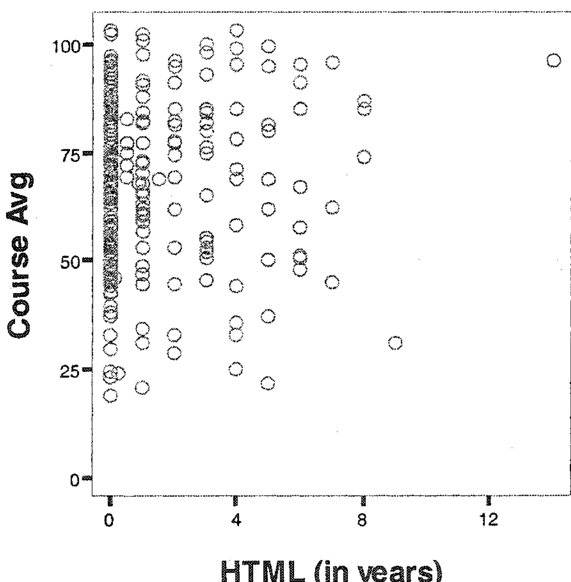


Course Avg by Basic (in years)

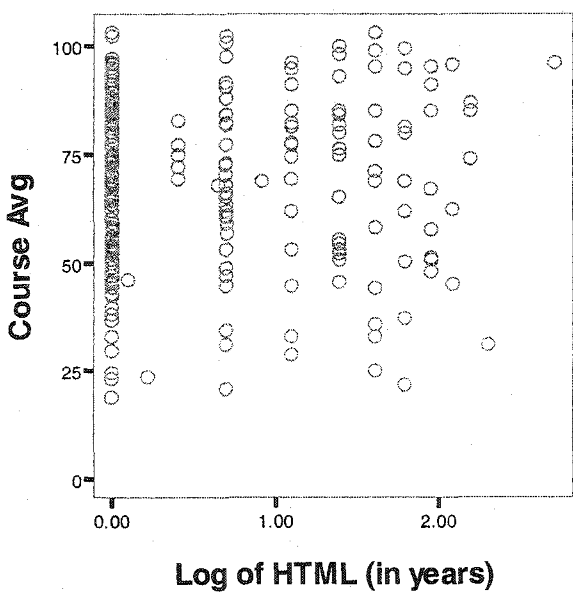


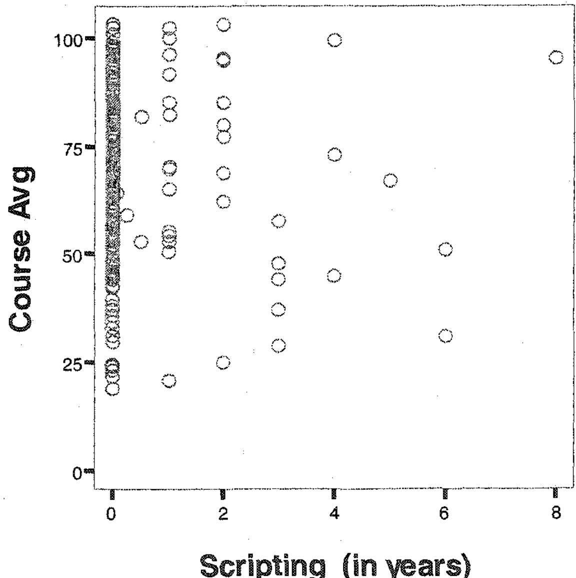
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.





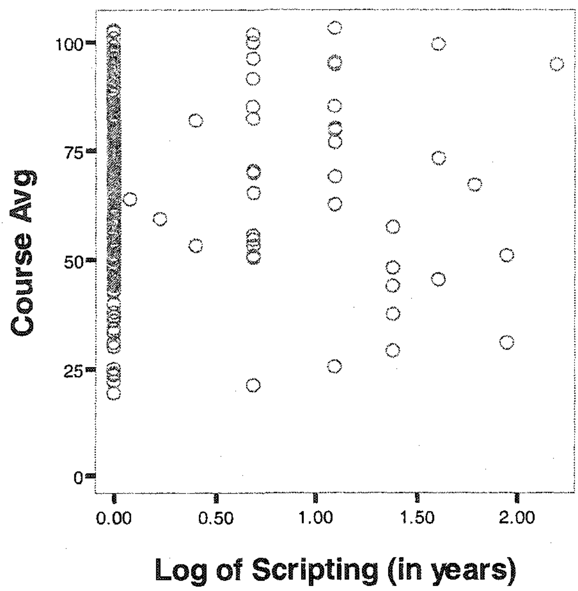
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.



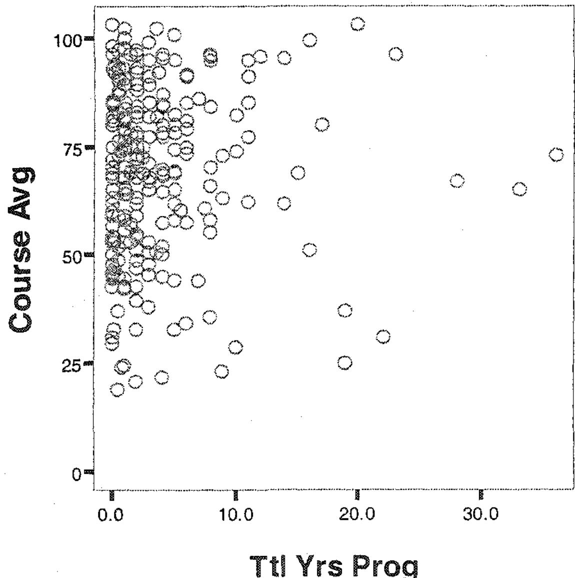


Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Course Avg by Log of Scripting (in years)

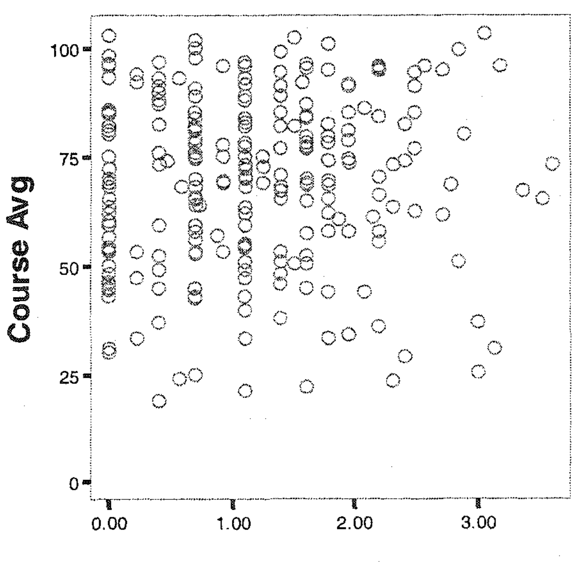


Course Avg by Ttl Yrs Prog

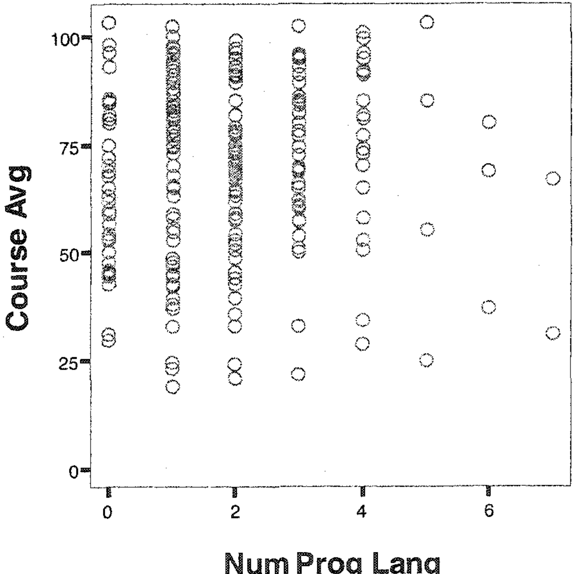


Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

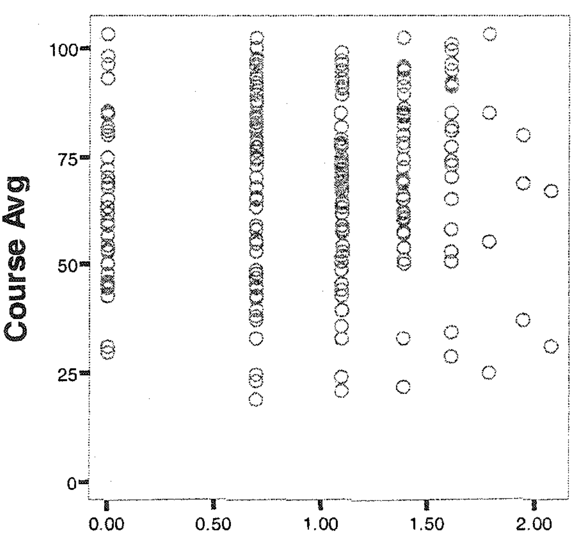
Course Avg by Log of Total Year Programming

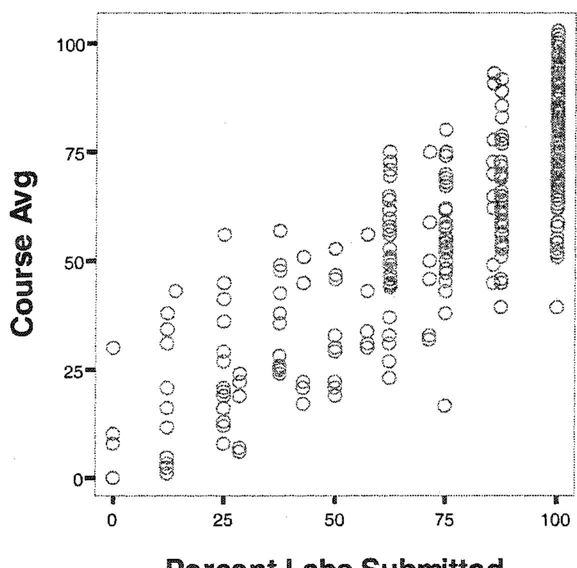


Course Avg by Num Prog Lang

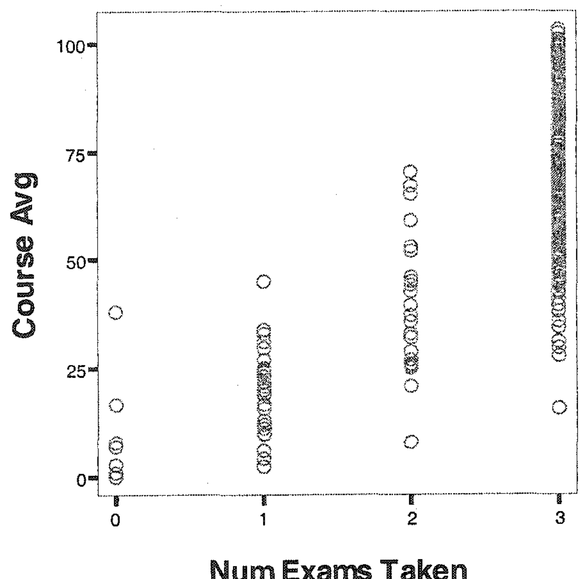


Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.





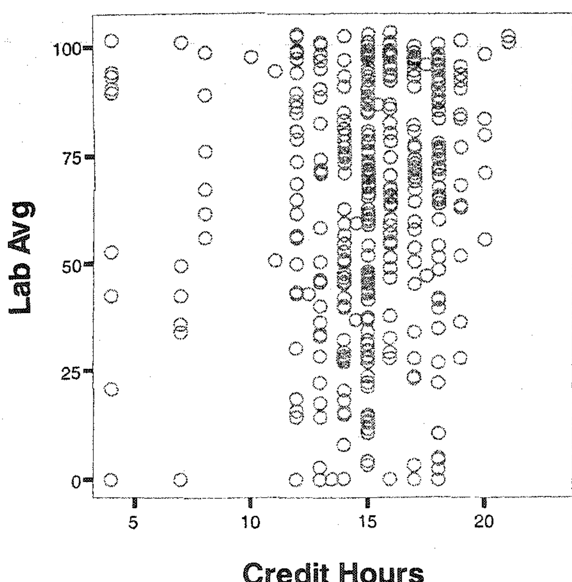
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.



Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

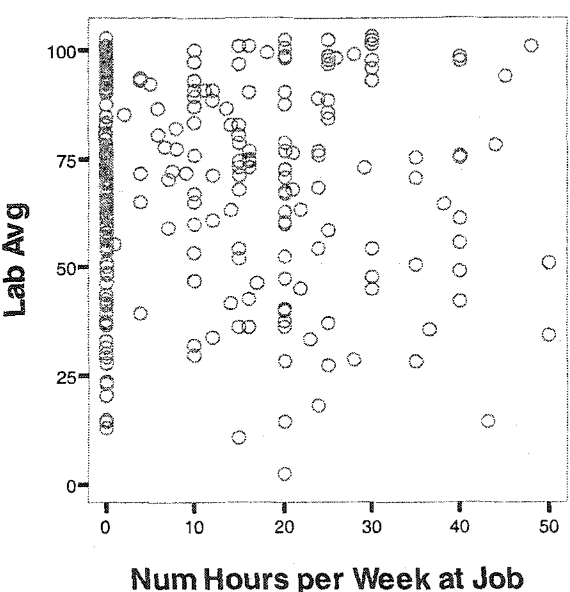
Appendix D

Lab Average Scatterplots

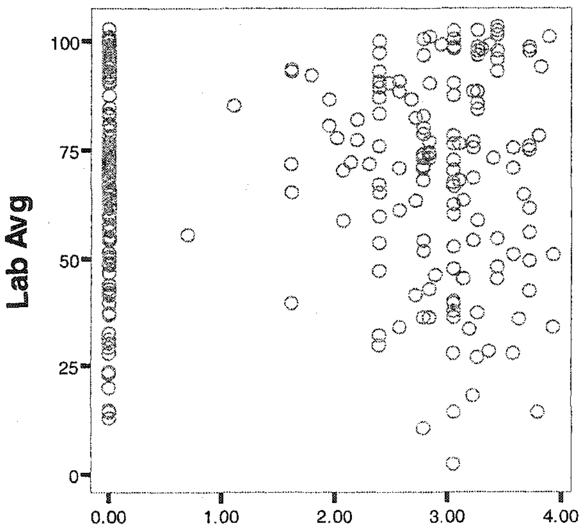


Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Lab Avg by Num Hours per Week at Job

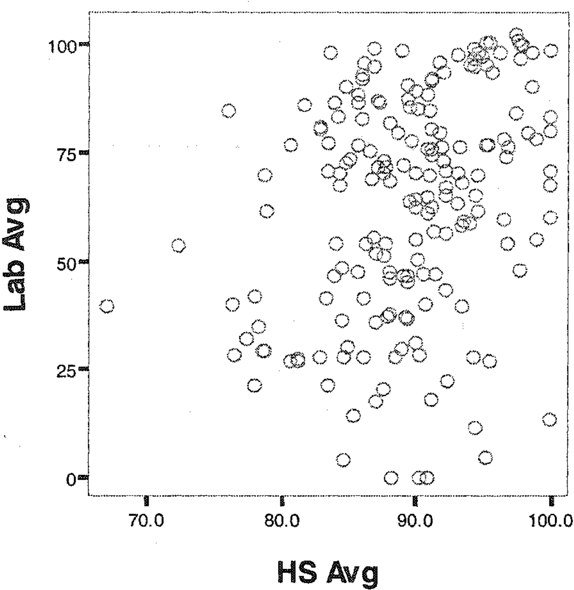


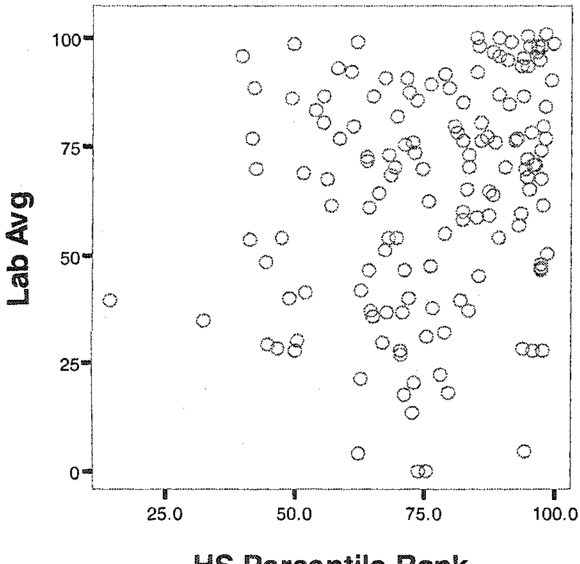
Lab Avg by Log of Number of Hours Worked at Job



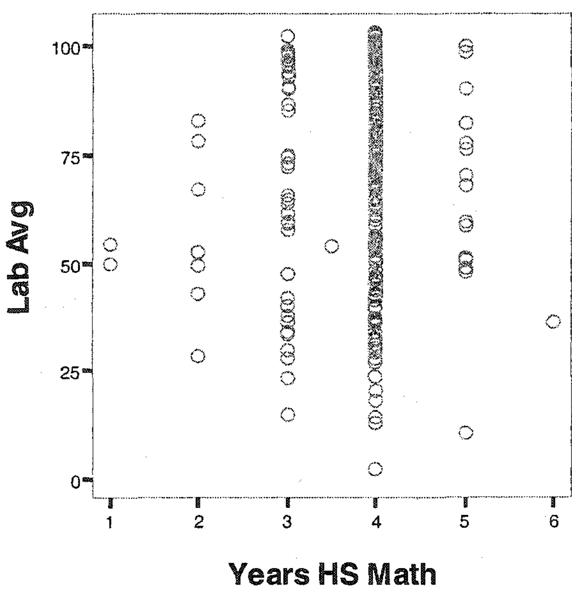
Log of Number of Hours Worked at Job

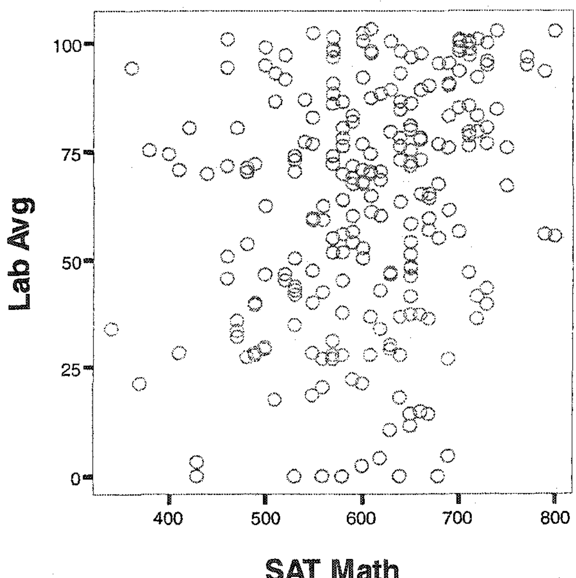
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.



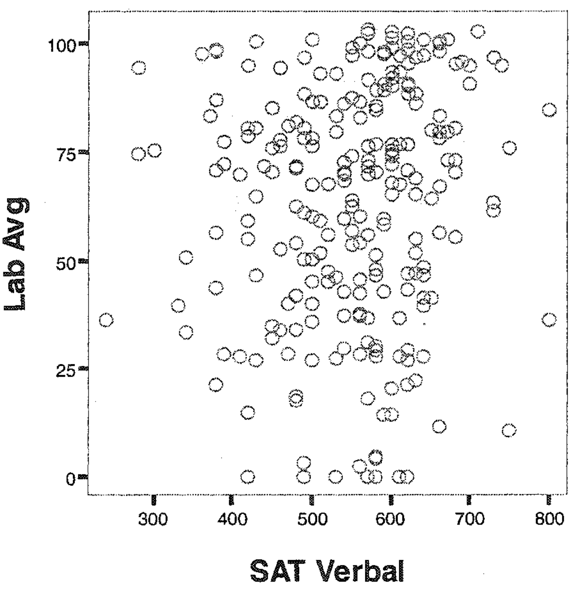


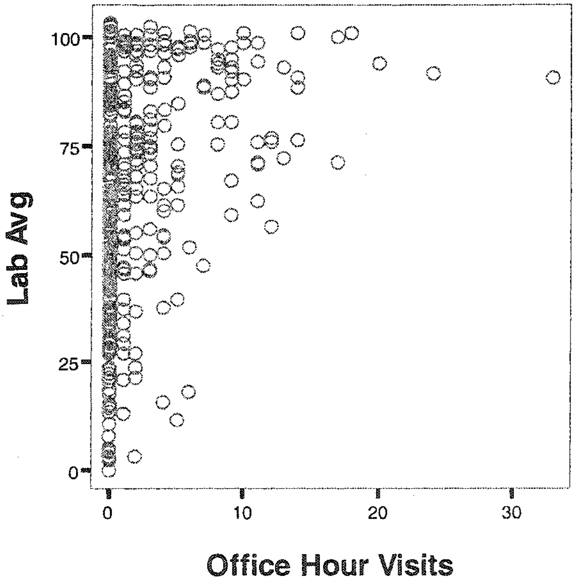
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.



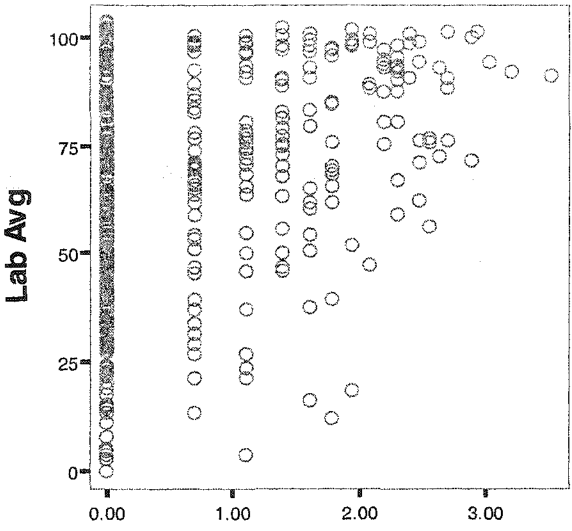


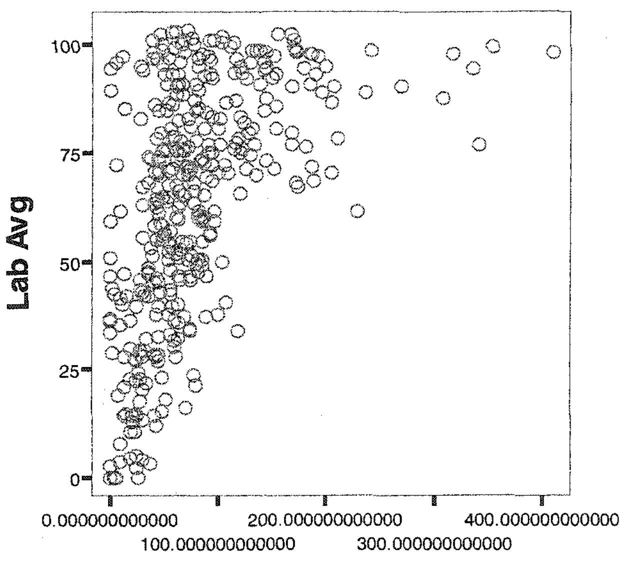
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.





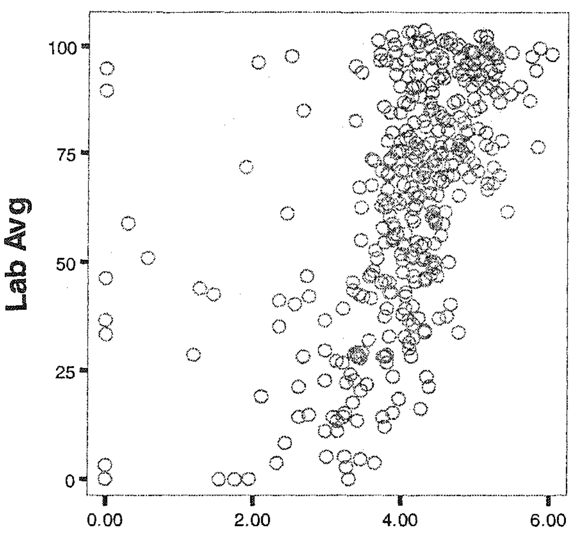
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.





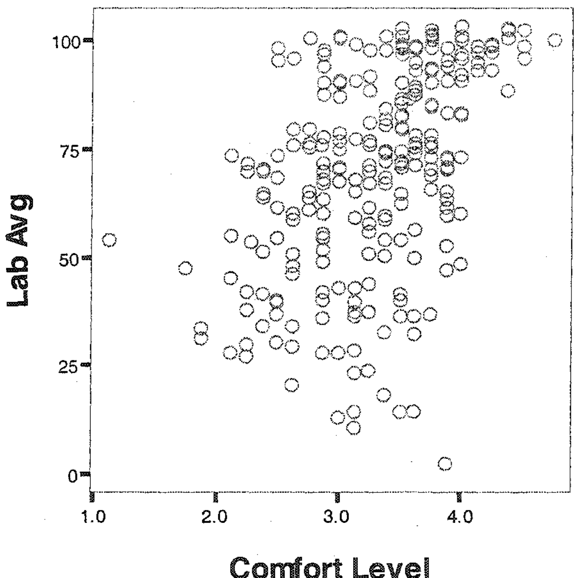
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Lab Avg by Log of Percent Recitation Time Usage



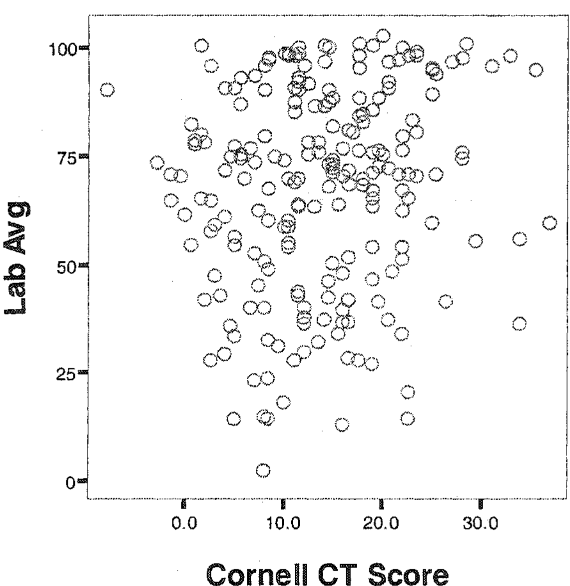
Log of Percent Recitation Time Usage

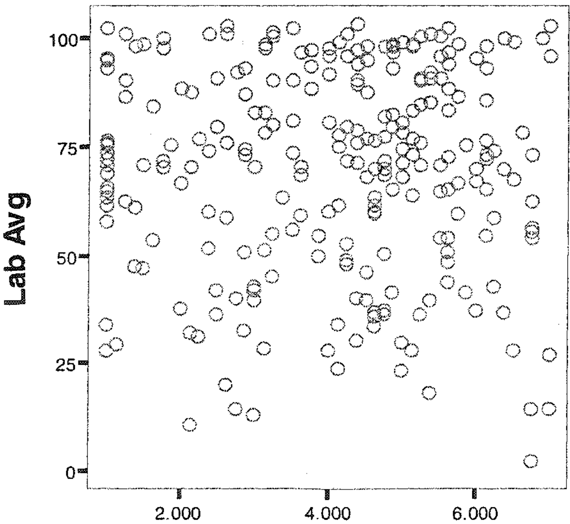
Lab Avg by Comfort Level



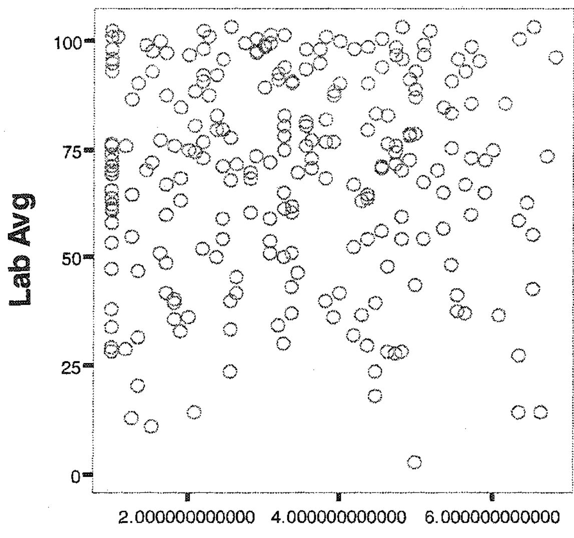
Comfort Level

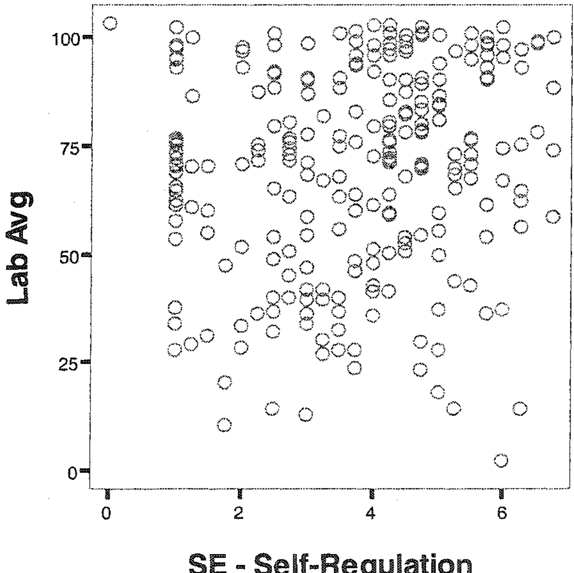
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.



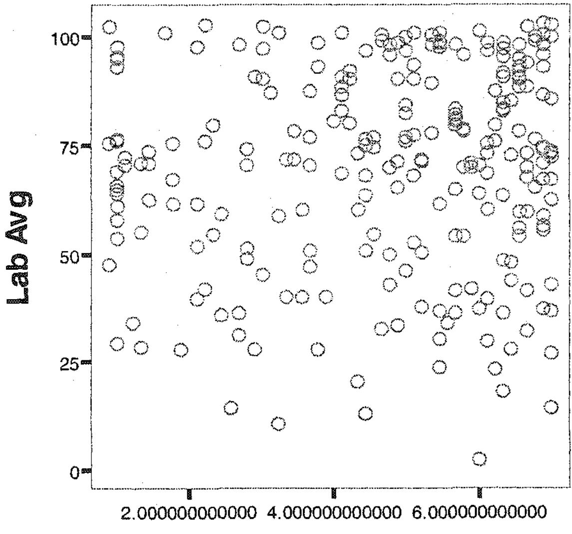


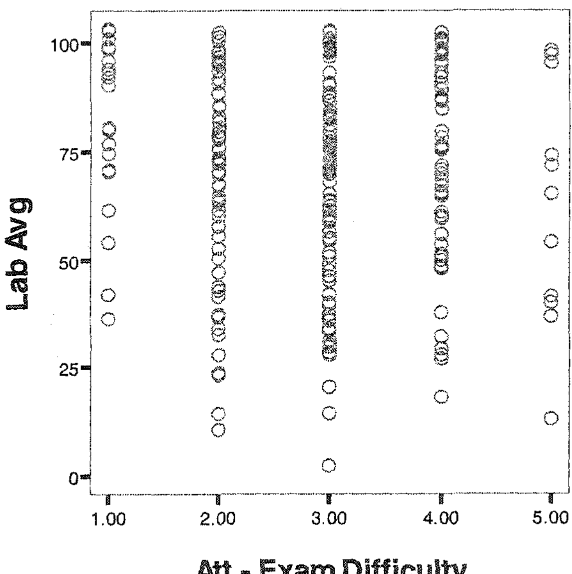
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.



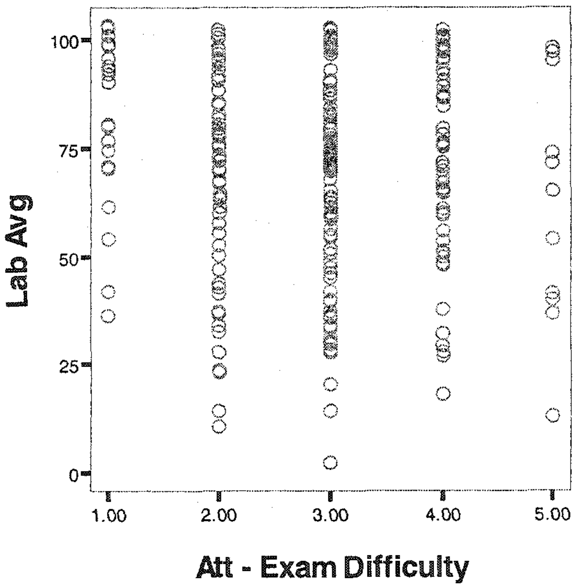


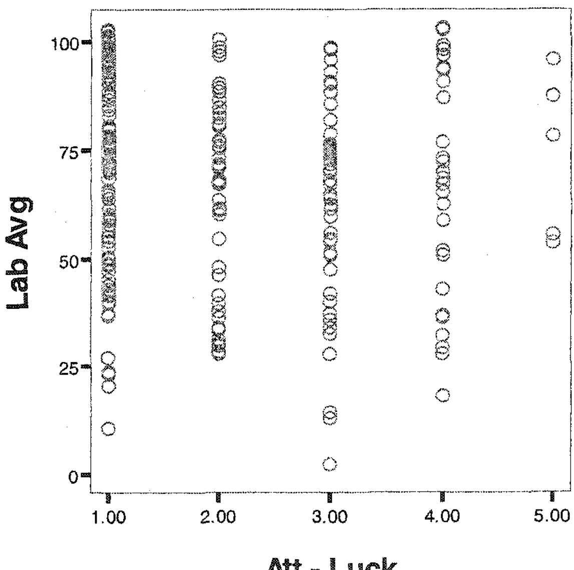
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.



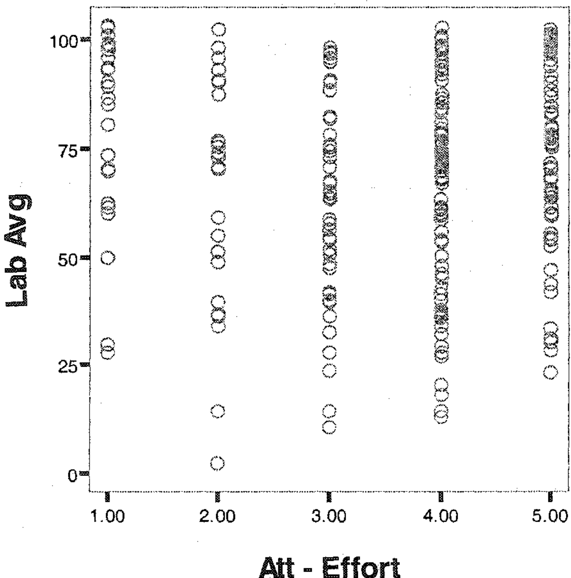


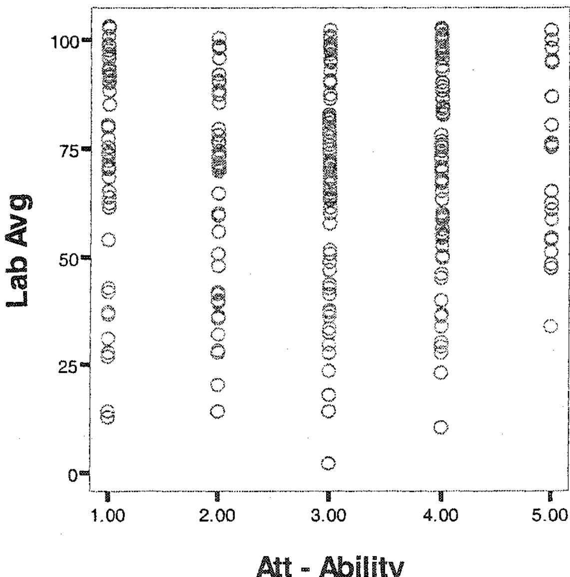
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.





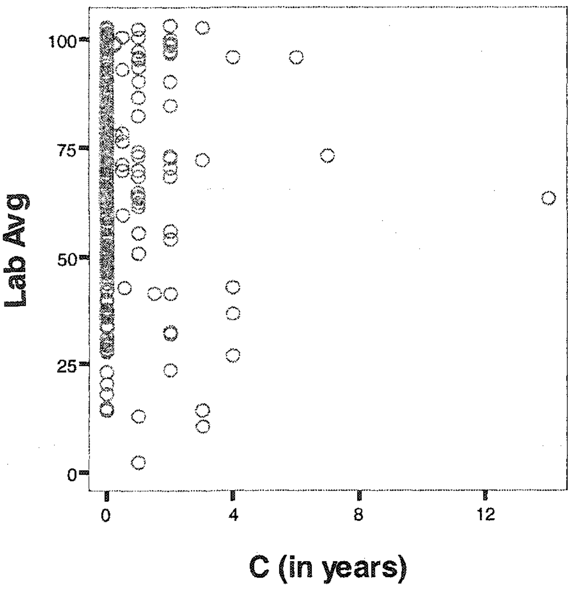
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.



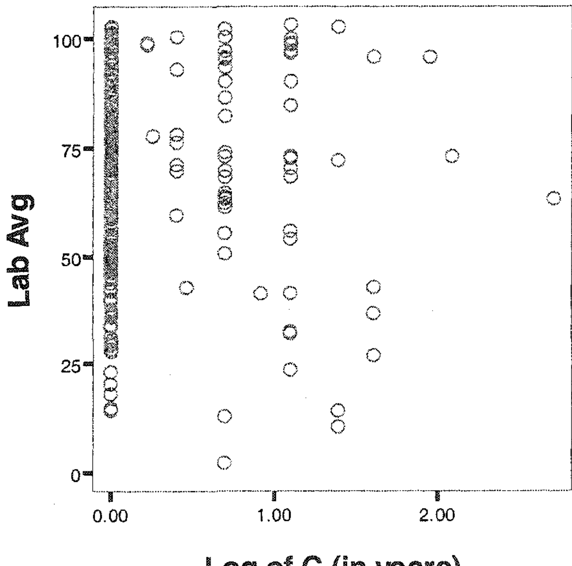


Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

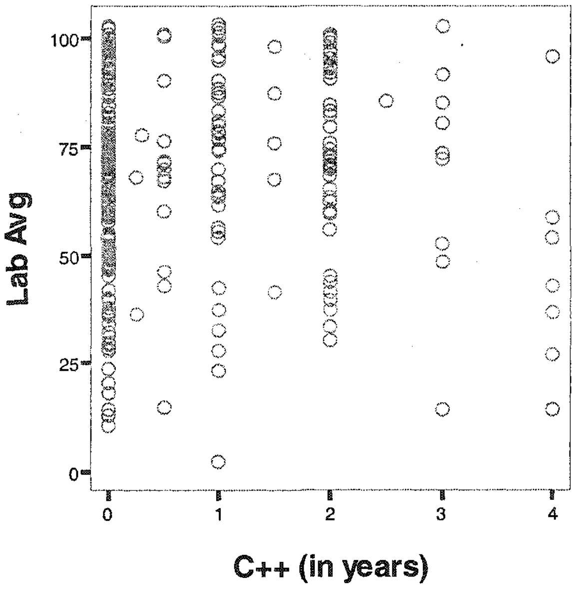
Lab Avg by C (in years)

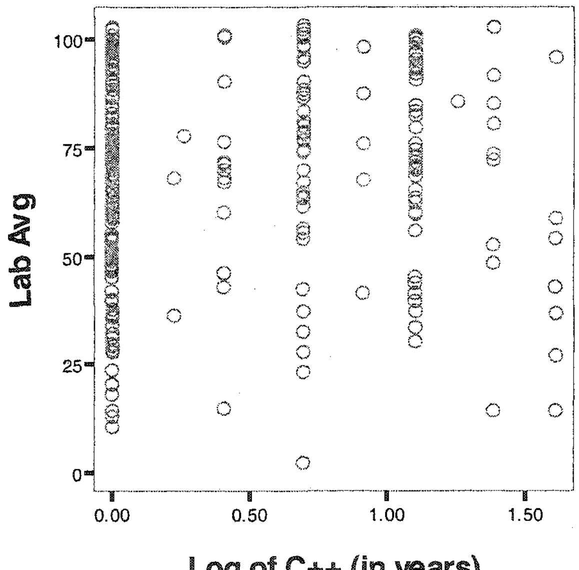


Lab Avg by Log of C (in years)

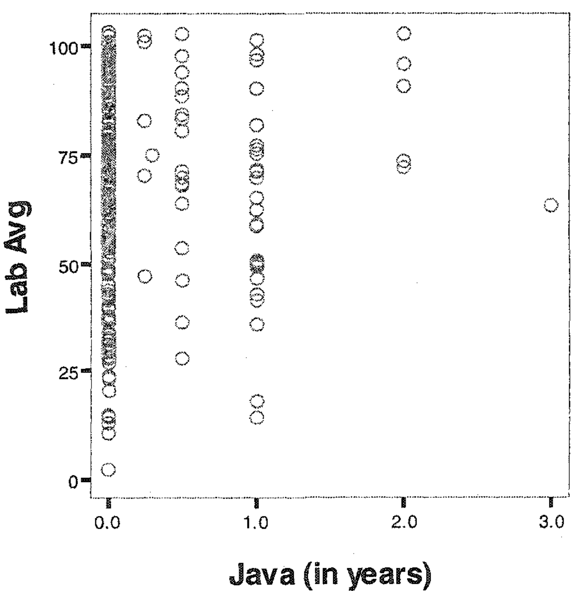


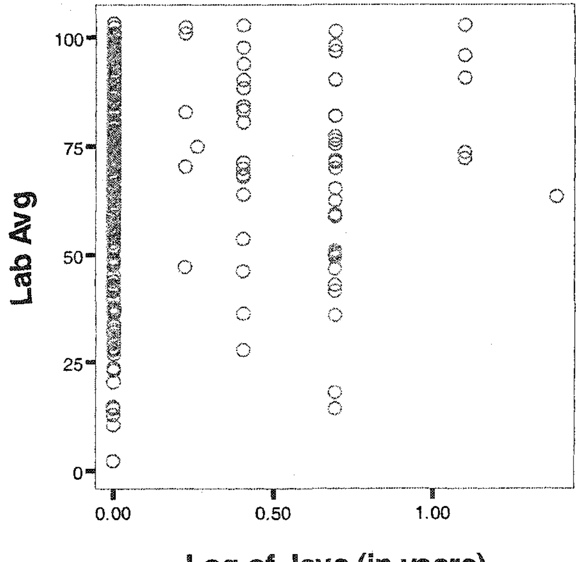
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.



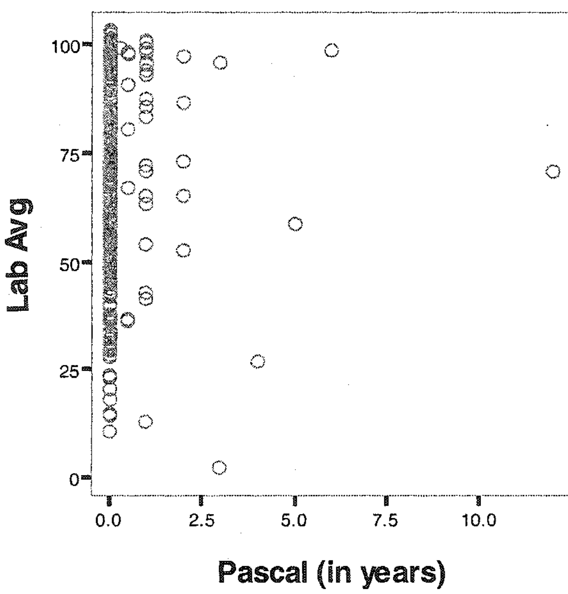


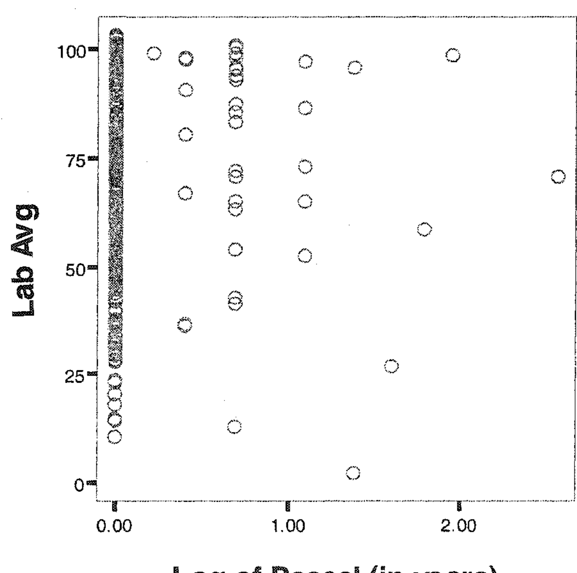
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.



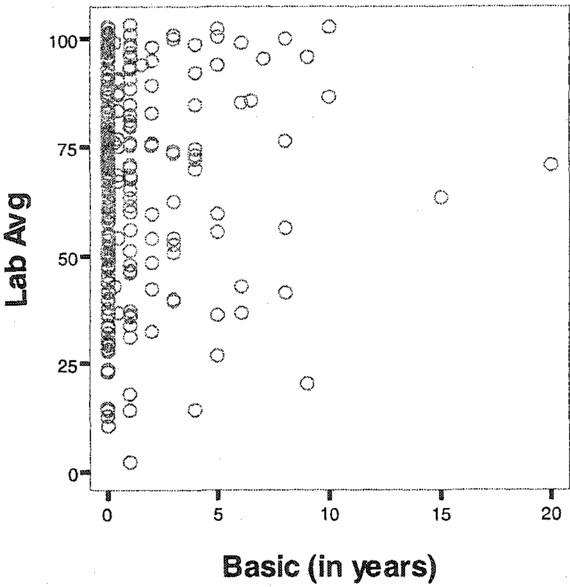


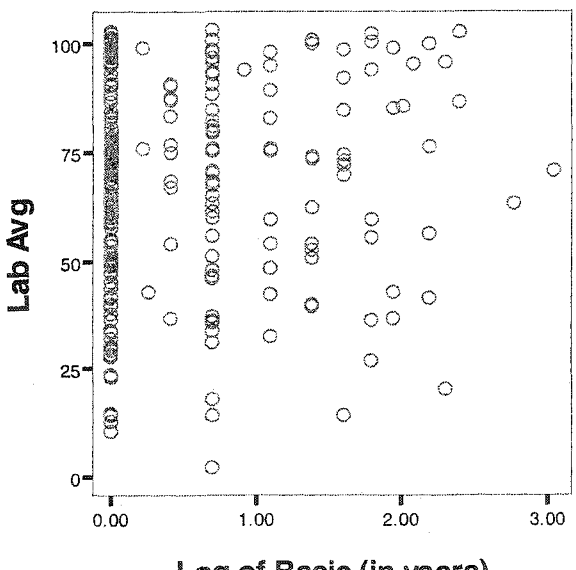
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.



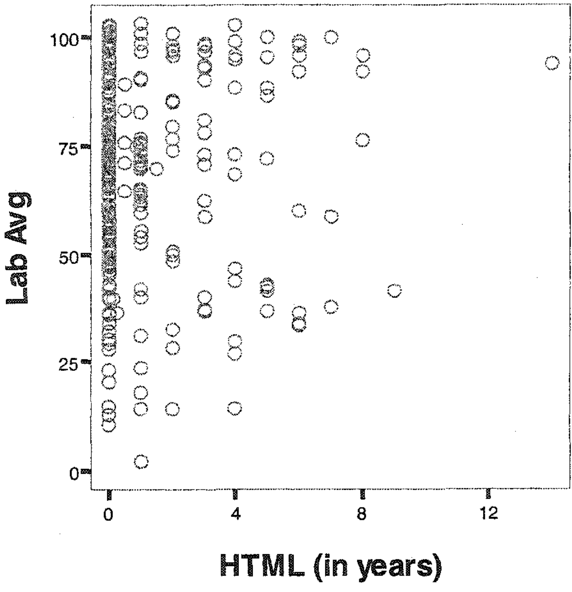


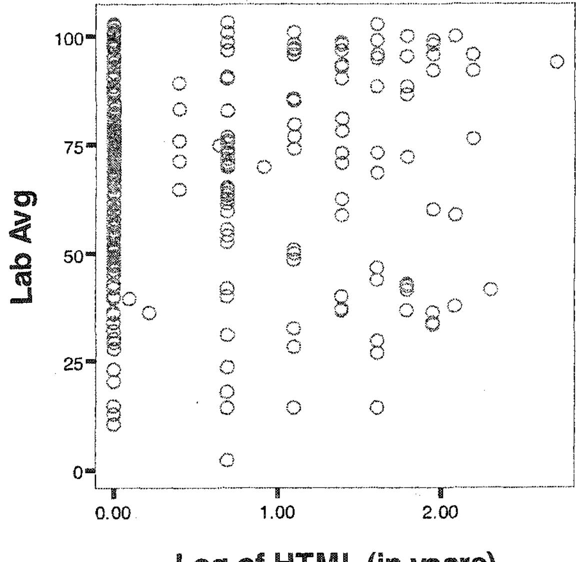
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.



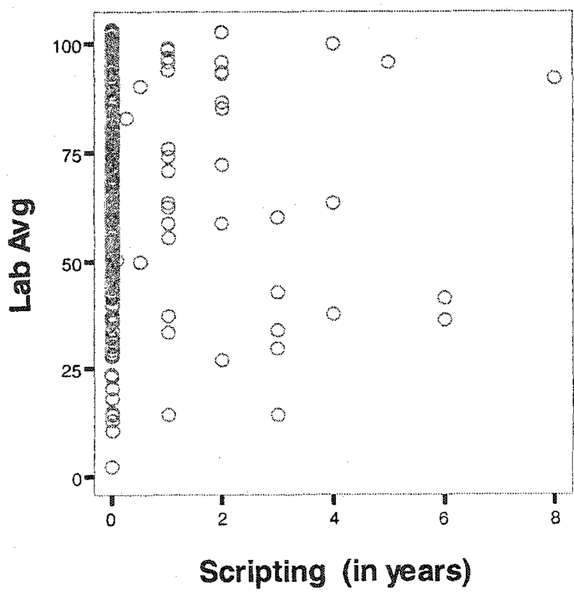


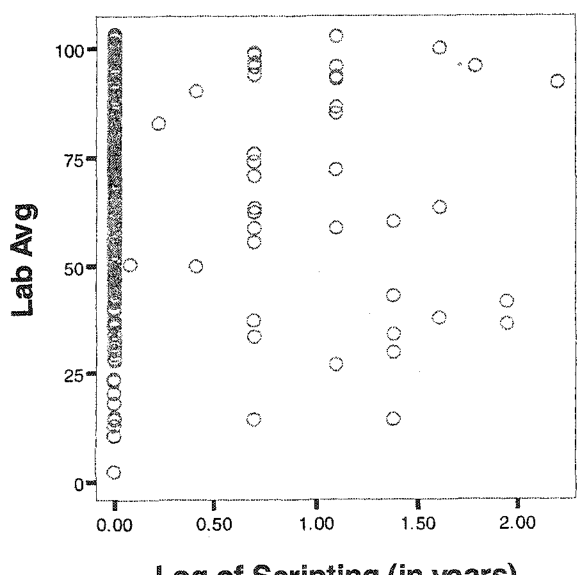
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.



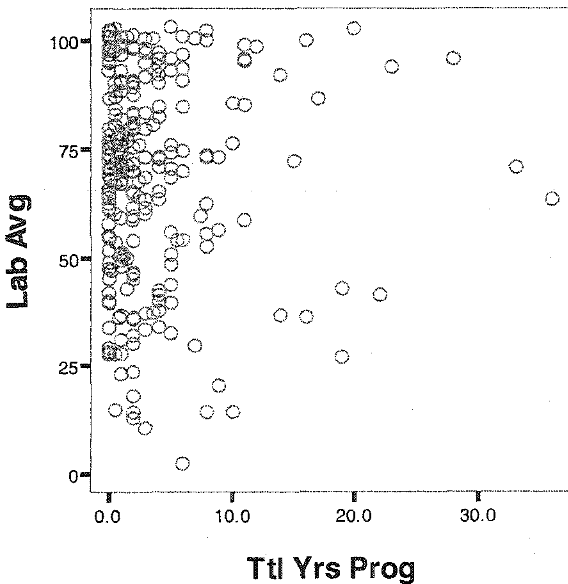


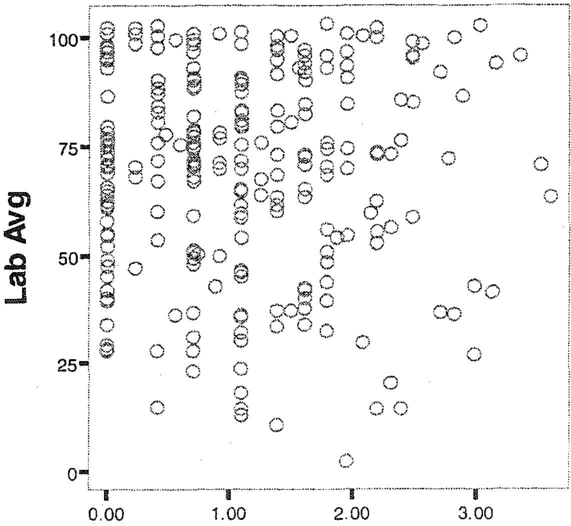
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.



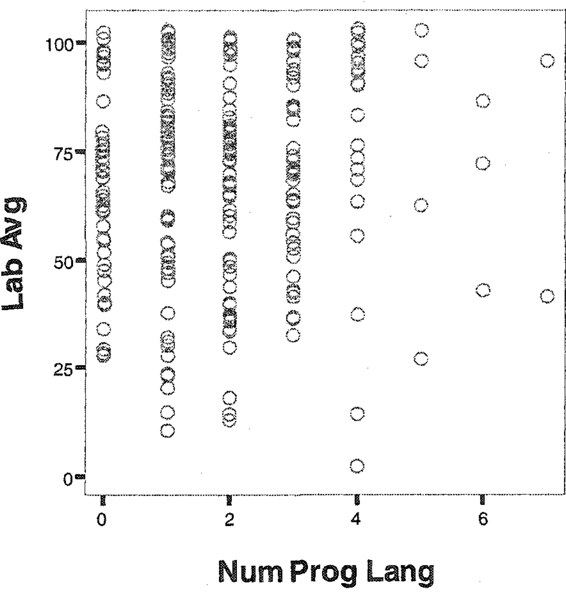


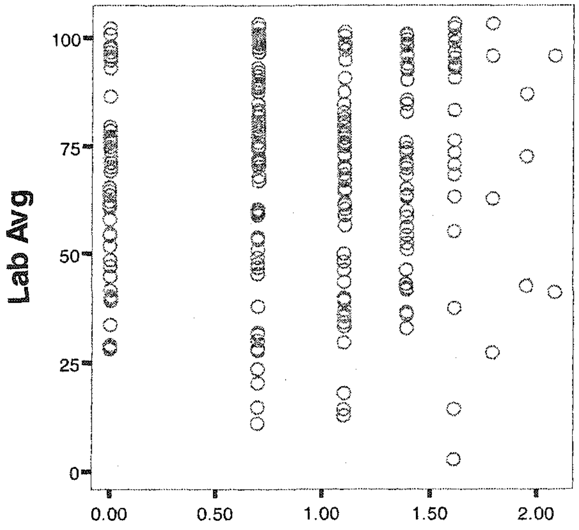
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.



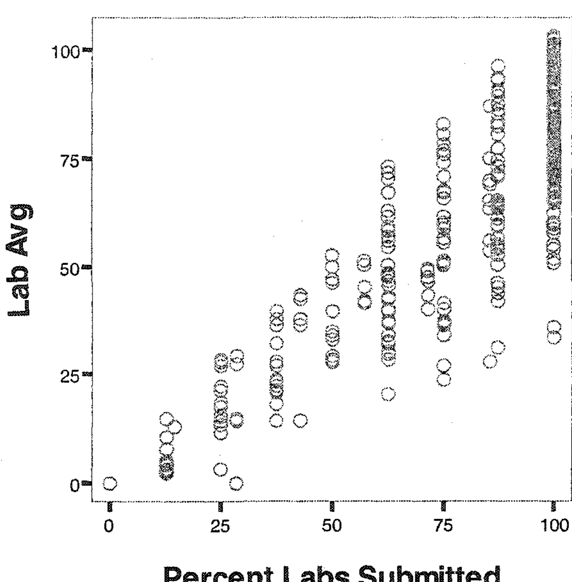


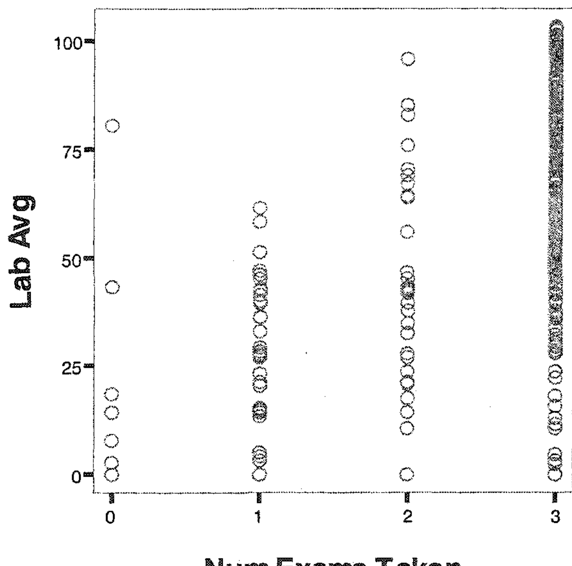
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.





Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

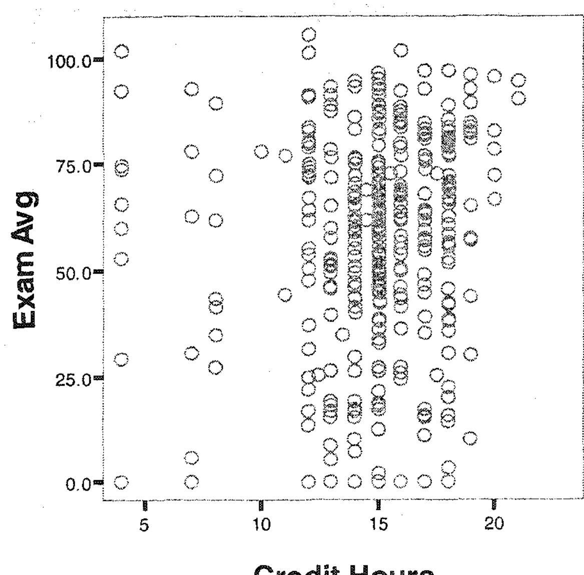




Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

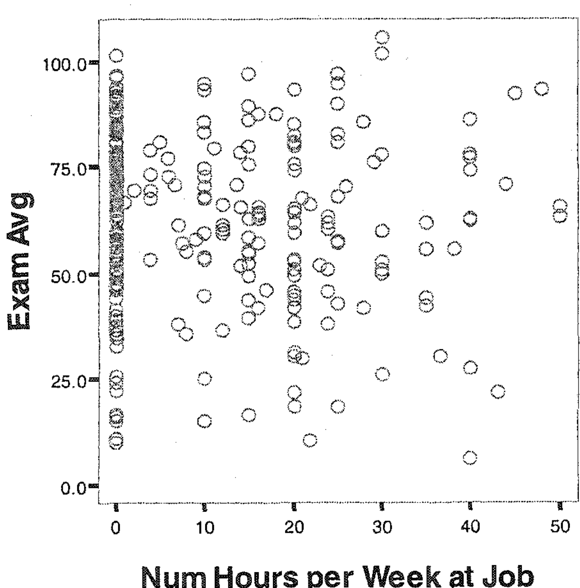
Appendix E

Exam Average Scatterplots

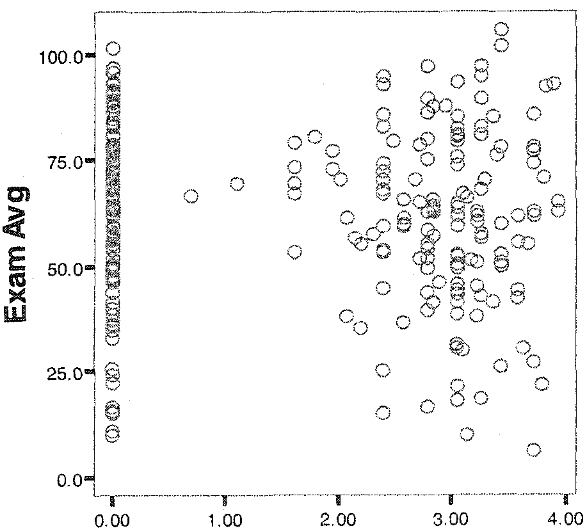


Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Exam Avg by Num Hours per Week at Job

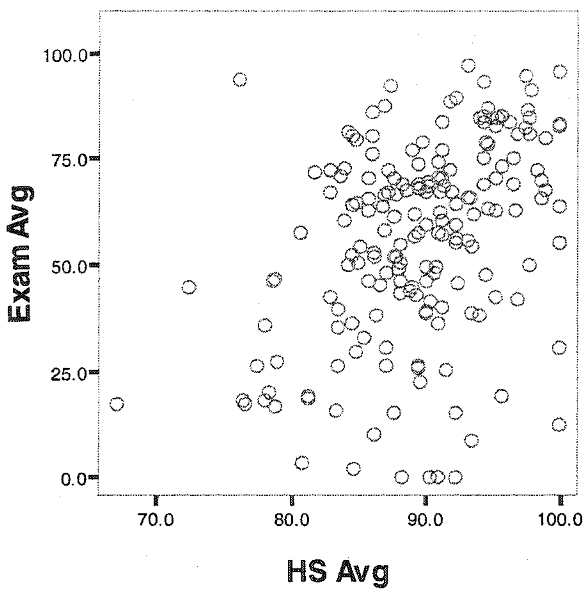


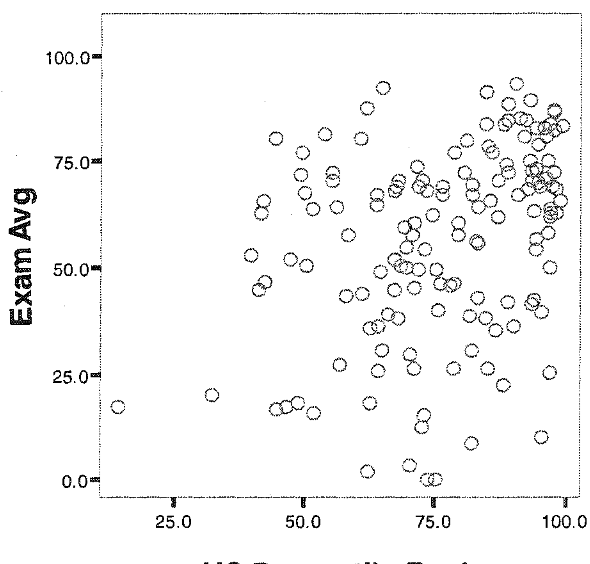
Exam Avg by Log of Number of Hours Worked at Job



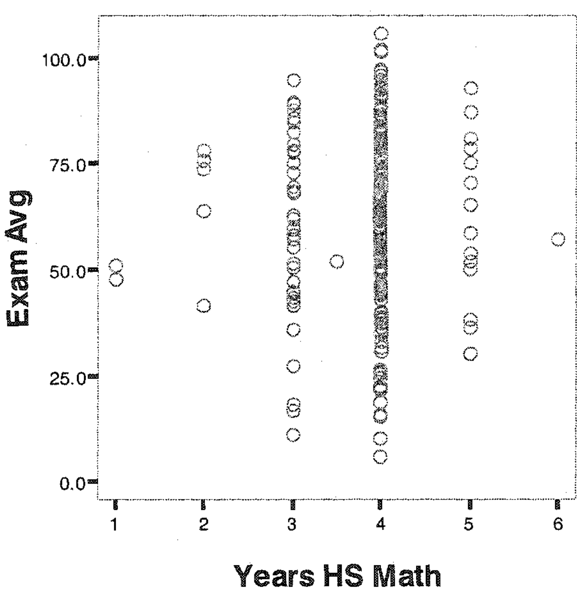
Log of Number of Hours Worked at Job

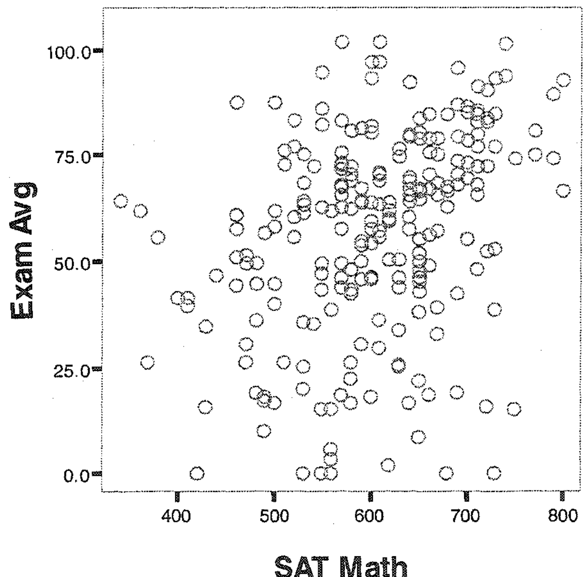
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.



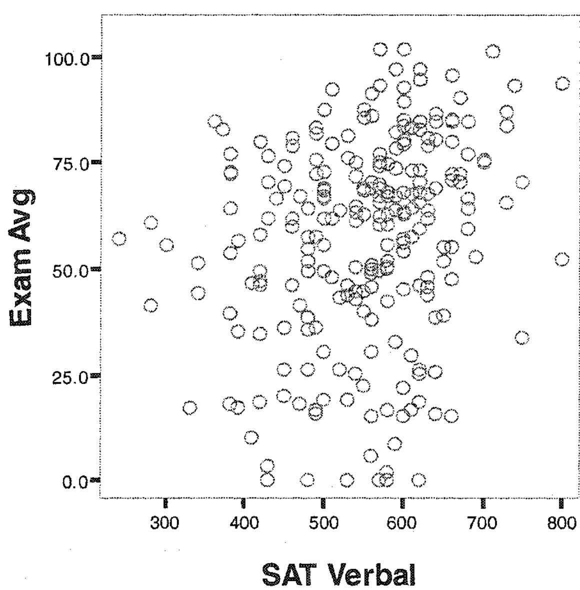


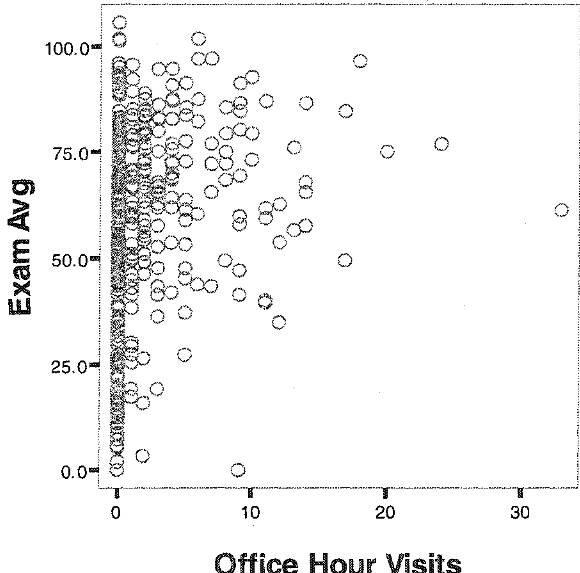
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.



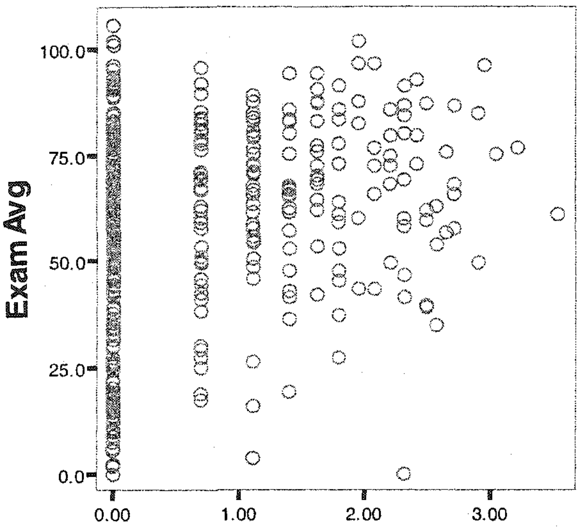


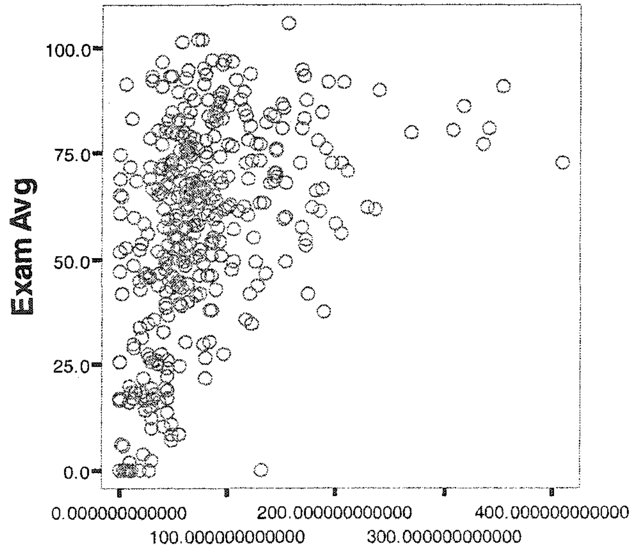
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.





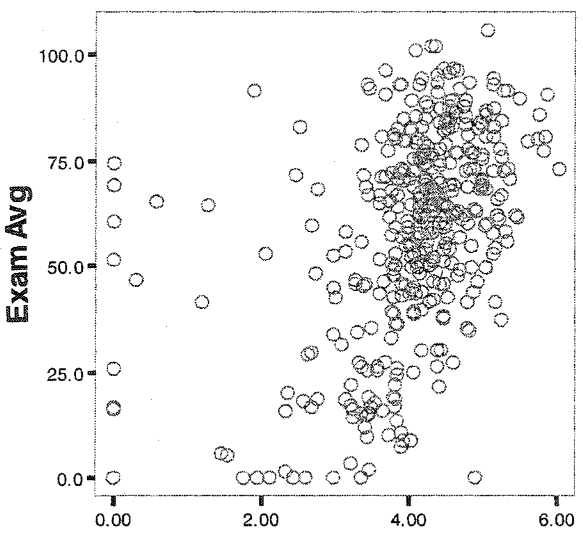
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.





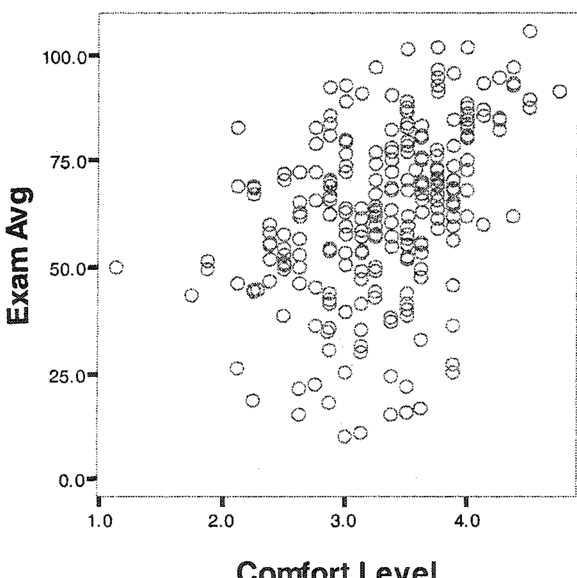
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Exam Avg by Log of Percent Recitation Time Usage

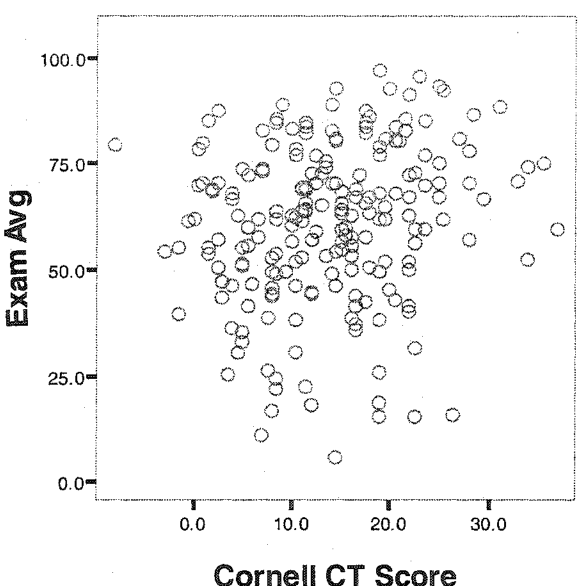


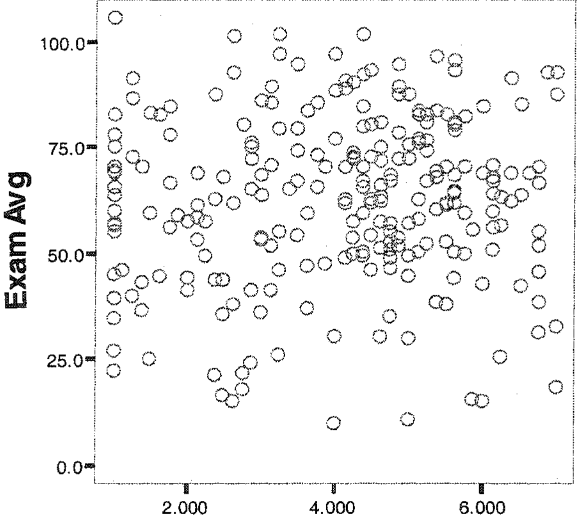
Log of Percent Recitation Time Usage

Exam Avg by Comfort Level

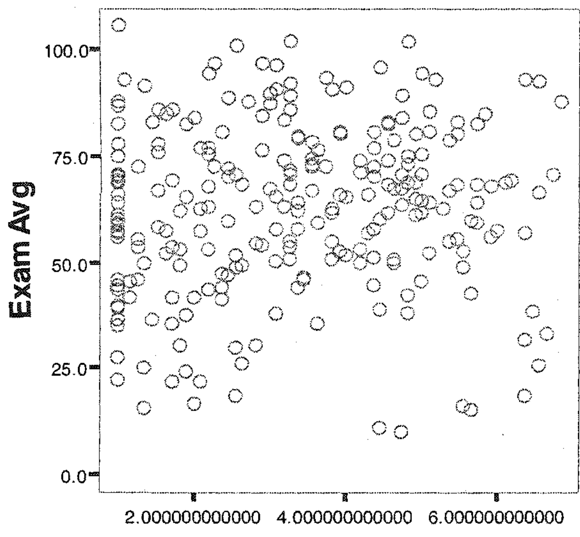


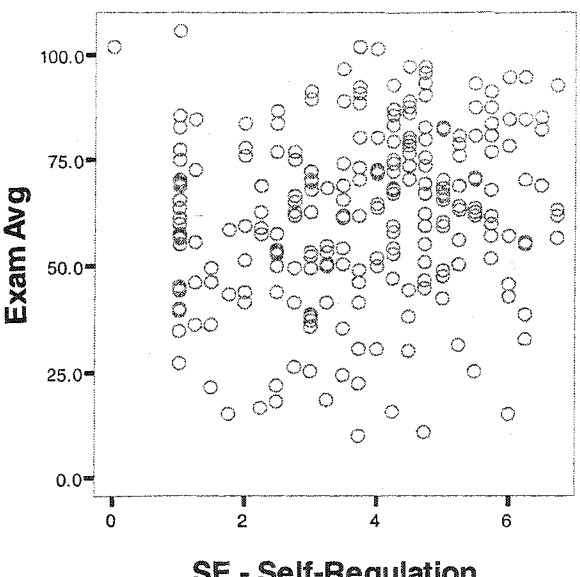
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.





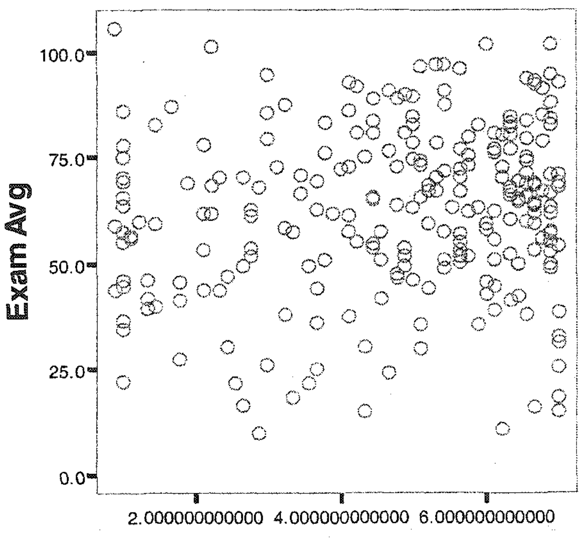
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.





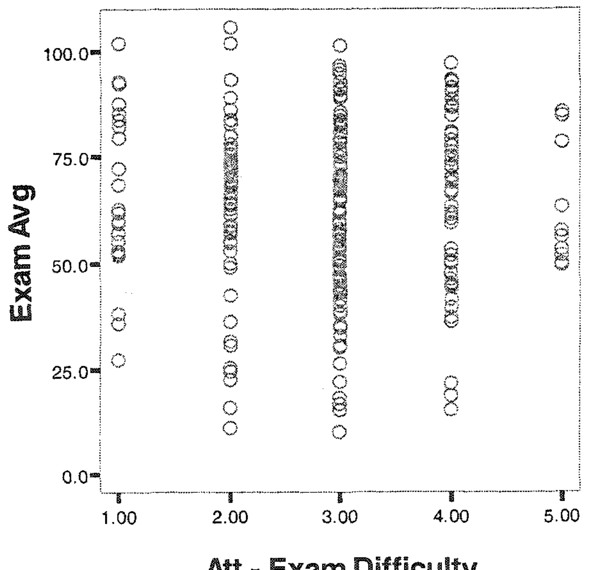
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Exam Avg by SE - Simple Programming



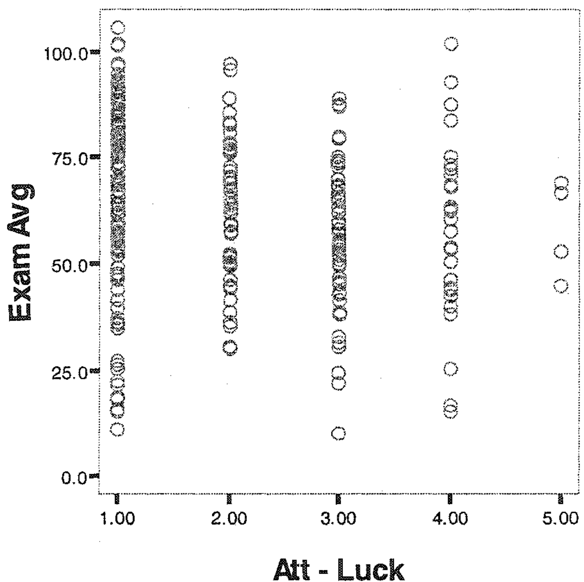
SE - Simple Programming

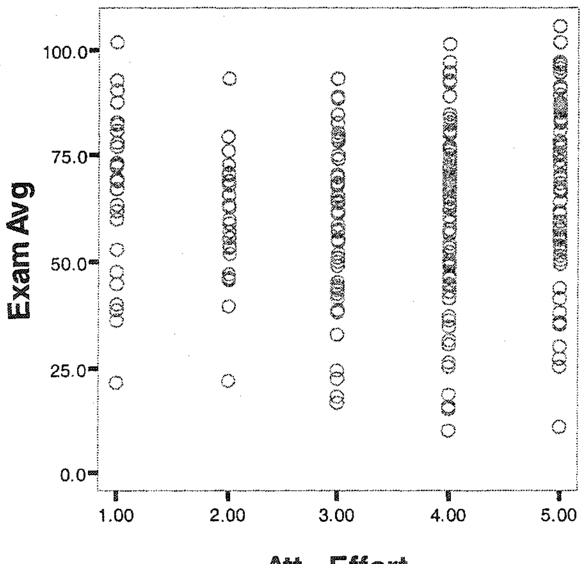
Exam Avg by Att - Exam Difficulty



Att - Exam Difficulty

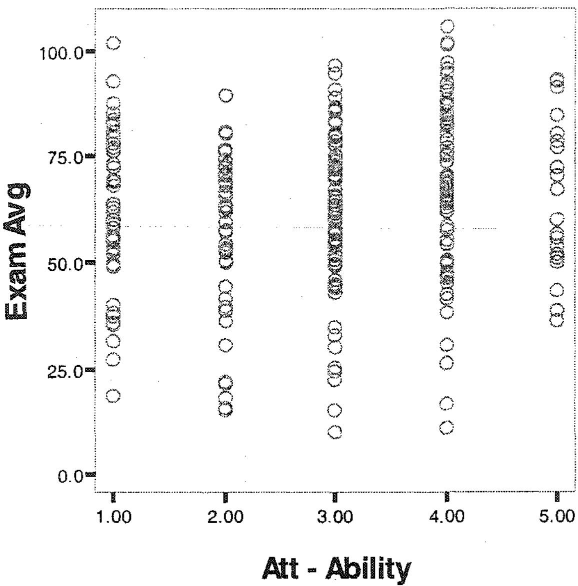
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.



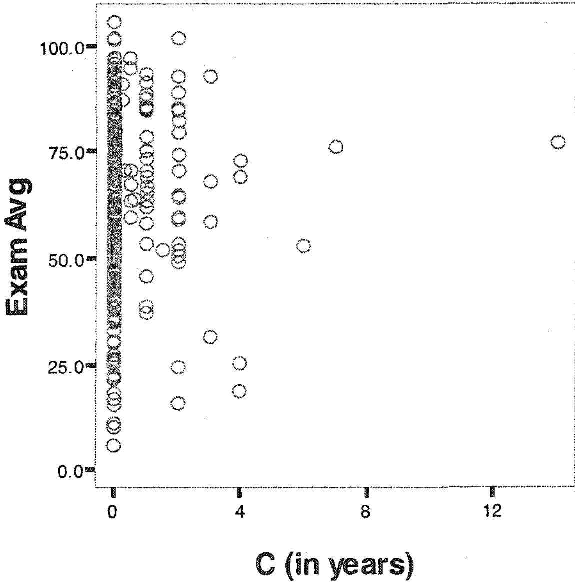


Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

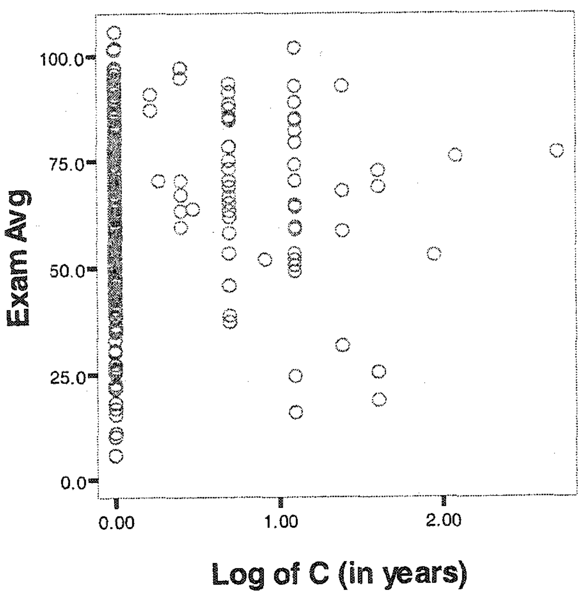
Exam Avg by Att - Ability

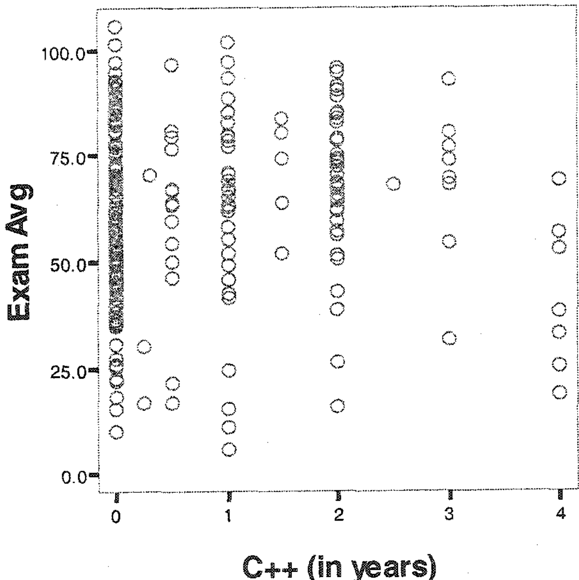


Exam Avg by C (in years)

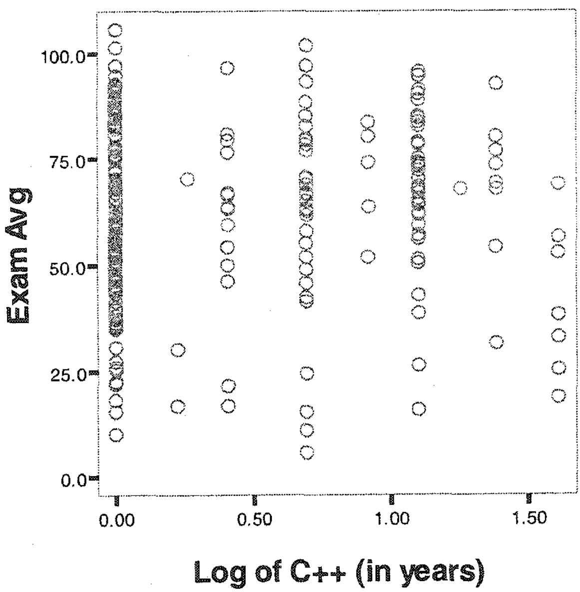


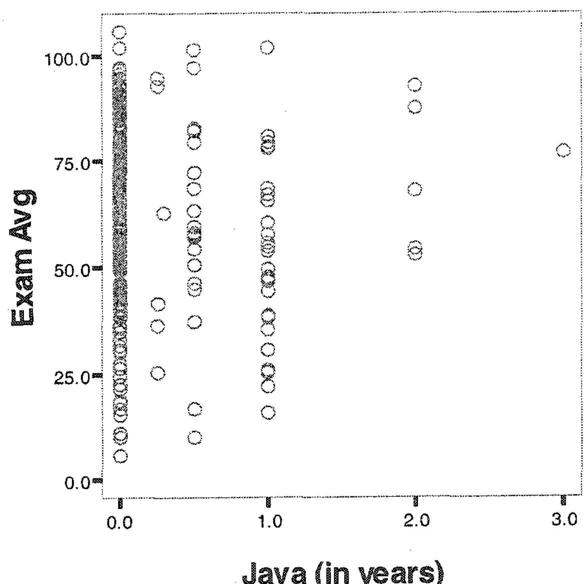
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.



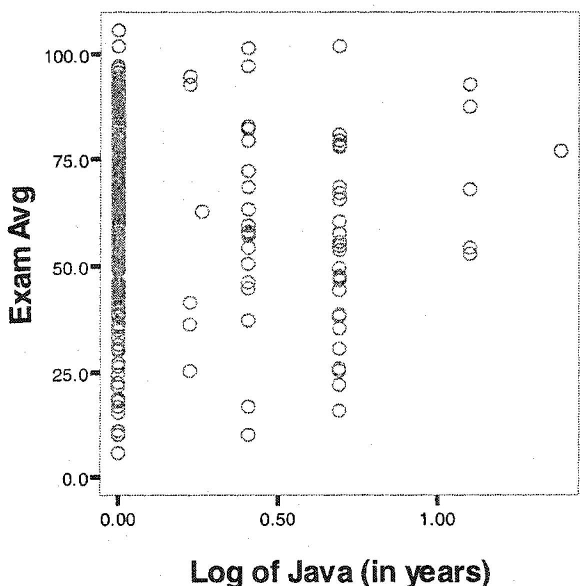


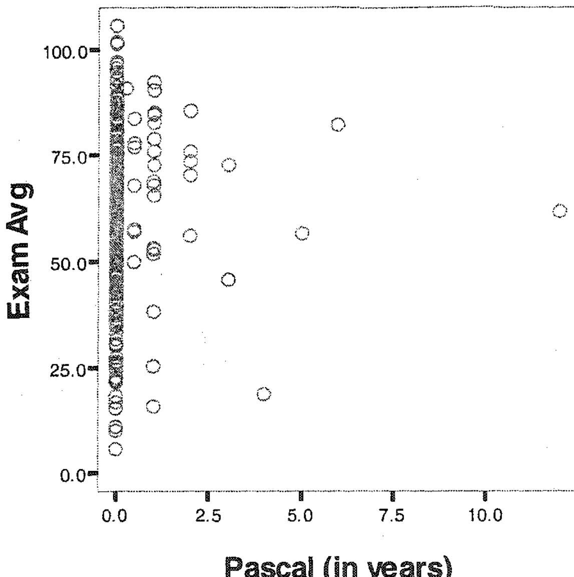
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.



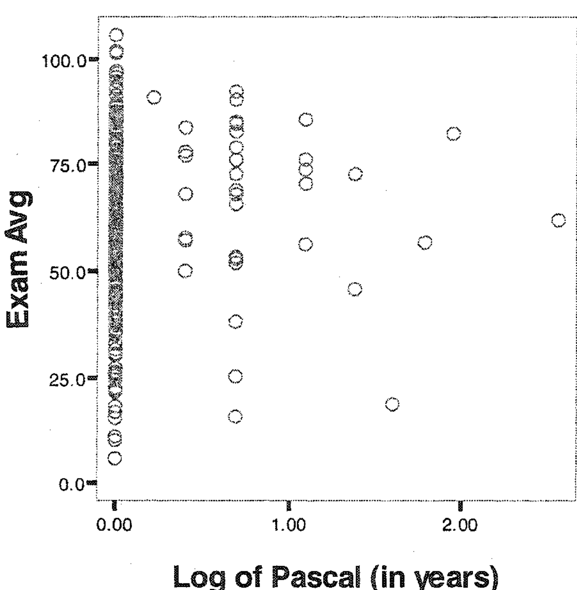


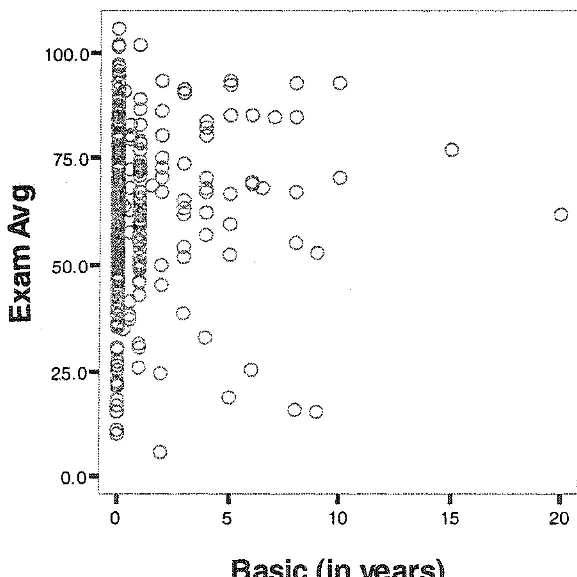
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.



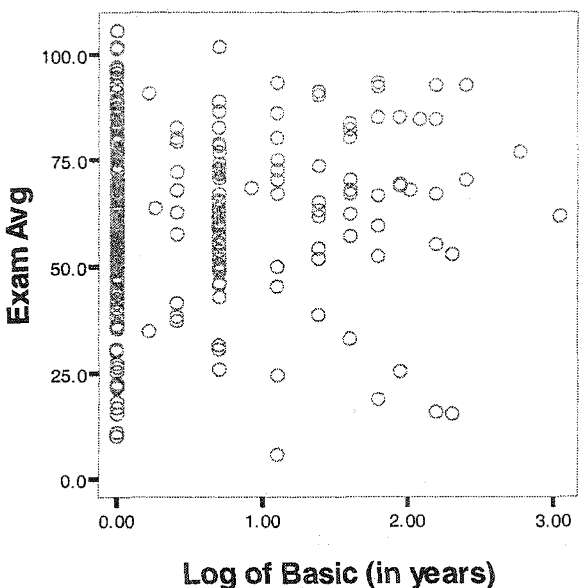


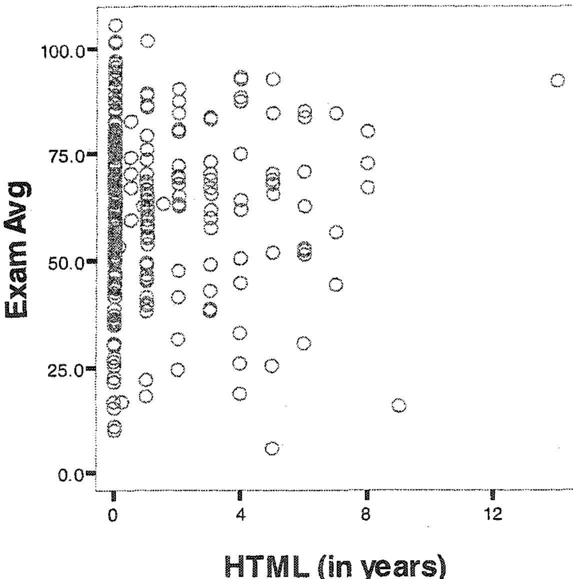
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.





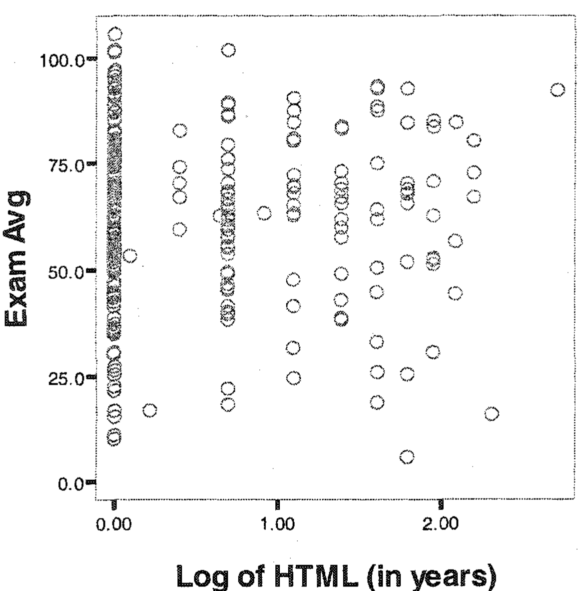
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.



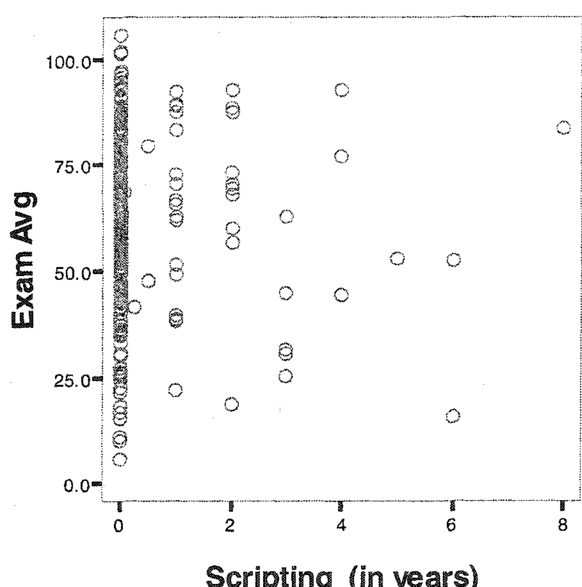


Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

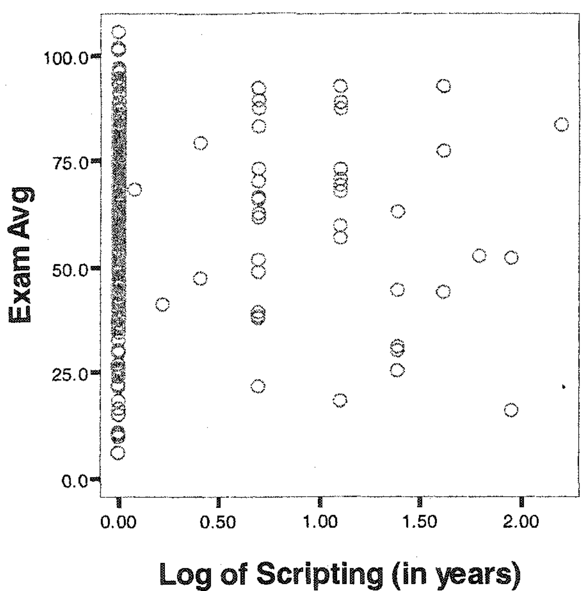
Exam Avg by Log of HTML (in years)

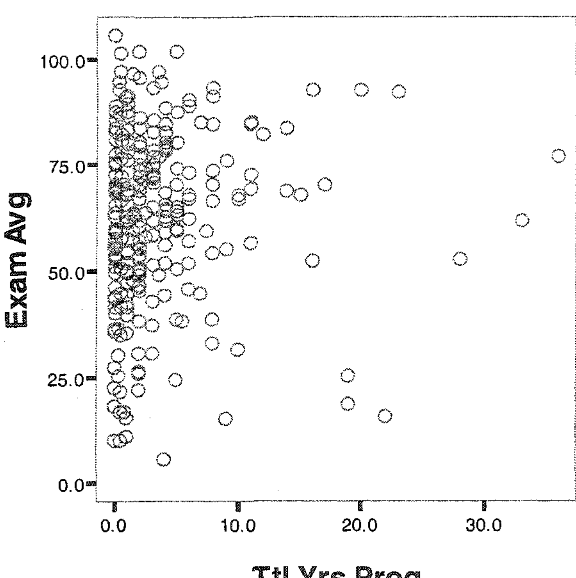


Exam Avg by Scripting (in years)

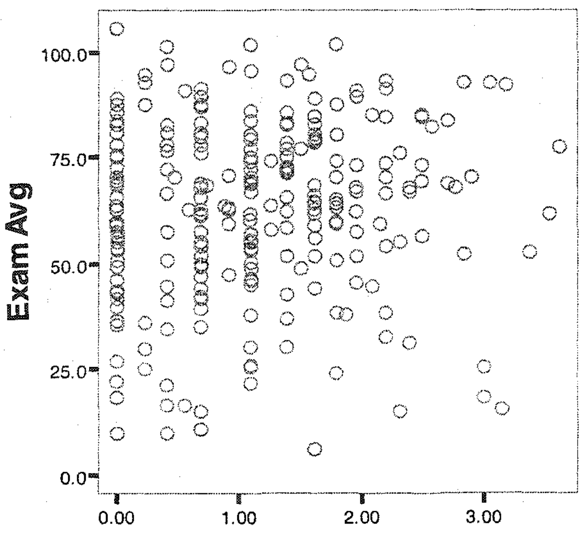


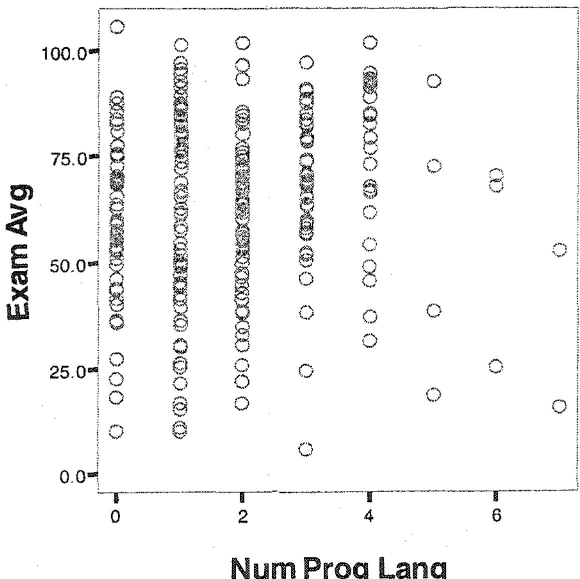
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.





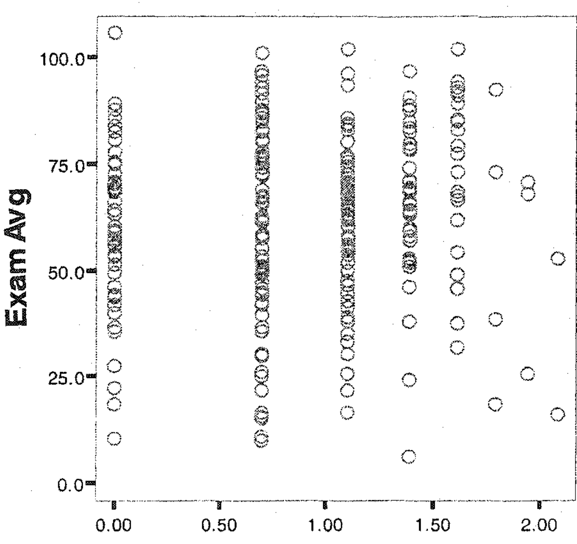
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.





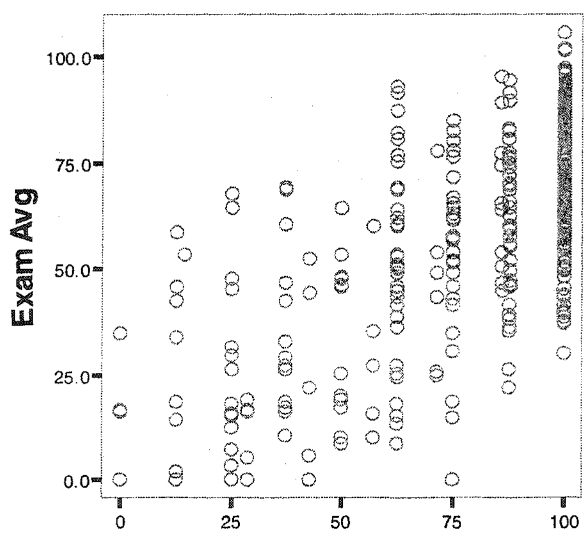
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.

Exam Avg by Log of Number of Programming Languages



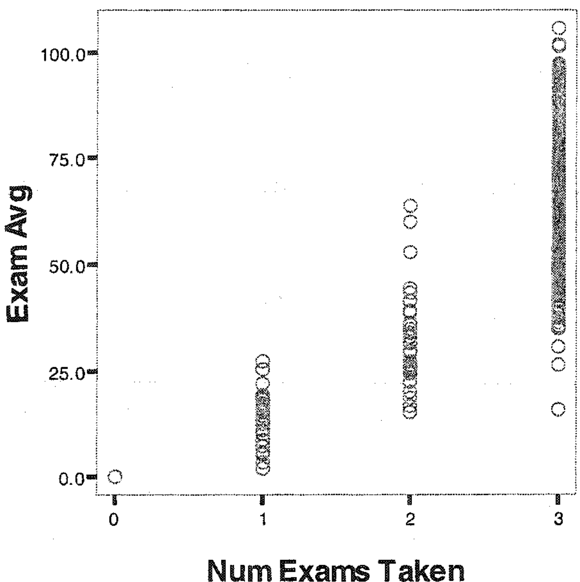
Log of Number of Programming Languages

Exam Avg by Percent Labs Submitted



Percent Labs Submitted

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.



Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.