

MEASURING THE EFFECTS OF LOW ASSISTIVE VS. MODERATELY ASSISTIVE
ENVIRONMENTS ON NOVICE PROGRAMMERS

by

EDWARD C. DILLON, JR.

MARCUS BROWN, COMMITTEE CHAIR

MONICA ANDERSON-HERZOG

SUSAN VRBSKY

PETER DEPASQUALE

CECIL ROBINSON

A DISSERTATION

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy
in the Department of Computer Science
in the Graduate School of
The University of Alabama

TUSCALOOSA, ALABAMA

2012

Copyright Edward Claudell Dillon, Jr. 2012
ALL RIGHTS RESERVED

ABSTRACT

Improving the novice's experience with programming has been an important research topic for some time. The high attrition rate of CS majors continues to be a problem. Incoming majors are being exposed to programming but many are driven away from the field.

As a way to engage novices with programming, many CS departments have adopted visual environments. However, not all novices are taught to program using visual environments. Typically, students are introduced to programming through either a visual or command line environment at the beginning stages of a CS curriculum.

The features in standard command line environments are not as assistive to programmers as visual environments. Novices must learn both language syntax and semantics while navigating the file system and compilation tools. On the other hand, visual environments with highly assistive features could constrict a novice to learn a fixed set of foundational programming skills that exclude exposure to syntax checking, compilation and file systems. Novices will eventually need to move to a less assistive environment to round out their skill set.

The objective of this research was to determine if certain environments are more appropriate for teaching novices how to program, based on their respective levels of feature assistance. There are anecdotally based motivations for using either tools with low assistive features like command line environments (promotes acquisition of useful mental models) or tools with moderate to high assistive features like visual environments (engages novices while programming). Unfortunately, no systematic study exists that supports either supposition.

This research was composed of three studies for evaluating environments with varying feature sets: a high school outreach, a CS1-Laboratory Study, and a CS1-Study. Engagement, comprehension, efficiency, and usability were used as measures to evaluate the environments during these studies. Overall, this research showed that a moderately assistive environment imposes a lower learning curve for novices, while a low assistive environment appears to broaden their understanding of programming.

DEDICATION

I dedicate this dissertation to the people who stood beside me during this journey. Particularly to my parents, Edward and Linda Dillon Sr., who instilled within me the discipline to achieve my goals along with their wisdom for making right decisions. To my sisters, Rosalind and Sheletha, for helping me develop the ambition to always strive to be the best. I must also thank the love of my life, Michaela, who is about to embark upon this journey, for her daily inspiration and encouragement. I want to thank all of my friends who saw the potential in me to do great things. Last but not least, I dedicate this dissertation to a special friend, Keona, who passed on but will never be forgotten.

LIST OF ABBREVIATIONS AND SYMBOLS

α	Alpha value or type-1 error; it represents the probability that a statistical test will give a false positive error
df	Degrees of freedom
<i>et al.</i>	And others
<i>etc</i>	Et cetera
<i>ex</i>	Example
H_0	Null hypothesis
H_a	Alternative hypothesis
H_{a_x}	Sub-hypothesis
<i>IDE</i>	Integrated Development Environment
p	P-Value; it represents the actual alpha value that arises from a statistical test
(s)	One or more objects that may be available
t	T-test symbol; it represents a statistical hypothesis test that follows a plotted distribution if the null hypothesis is true
<i>vs</i>	Versus
χ^2	Chi-squared symbol
$\&$	Ampersand
*	Asterisk/Multiplication sign
=	Equal to
#	Numerical symbol

- < Less than
- ~ Roughly equal to
- % Percent sign

ACKNOWLEDGMENTS

I would like to thank my advisor and committee chair, Dr. Marcus Brown, for encouraging me to do research in this area. I would also like to thank one of my undergraduate mentors from the University of Mississippi, Dr. Dawn Wilkins, who introduced me to this area of research. A special thanks to Dr. Monica Anderson-Herzog, who not only helped me with conducting this research, but served as a mentor, role model, and a guide during this process. To Dr. Paul Mohr and Alabama Commission of Higher Education, I thank you for providing me with funding while pursuing a Ph.D. To Dr. Ansley Abraham, Ms. Tammy Wright, and the SREB family, I thank you for your support, hospitality, and encouragement. I would like to thank Dr. Viola Acoff, Dr. Louis Dale, and the LSAMP-Bridge to the Doctorate Program for preparing me as a Masters student for the journey into the Ph.D. To Dr. Donald Cole, Demetria Hereford and the Ronald E. McNair program, I thank you for exposing me to the idea of graduate school. To Ms. Jacqueline Vinson and the IMAGE program, who also influenced me to continue my education at the graduate level, I thank you as well. To Dr. Marcus Ashford and Dr. Karen Torres, I would like to thank the both of you for serving as my Tide Together mentors. I would also like to thank my committee members, Dr. Susan Vrbsky, Dr. Peter DePasquale, and Dr. Cecil Robinson for agreeing to be on my committee and also providing helpful comments and questions to improve my research. I would like to thank all of my fellow colleagues in the CS department as well as other departments who served as motivators and great friends during

my endeavor. Most of all, I would like to pay homage to my Heavenly Father for leading and guiding me through the good as well as challenging times during this journey.

CONTENTS

ABSTRACT.....	ii
DEDICATION.....	iv
LIST OF ABBREVIATIONS AND SYMBOLS.....	v
ACKNOWLEDGMENTS.....	vii
LIST OF TABLES.....	xvi
LIST OF FIGURES.....	xxi
1. INTRODUCTION.....	1
1.1 Contributions.....	3
1.2 Research Hypotheses.....	5
1.3 Manuscript Outline.....	6
2. LITERATURE REVIEW & BACKGROUND.....	7
2.1 Paradigms.....	8
2.1.1 OO vs. Non-OO Programming.....	8
2.1.2 Concepts in a CS1 Course.....	9
2.2 Languages.....	10
2.3 Programming Environments - Overview.....	12
2.3.1 Specialized Applications.....	12
2.3.1.1 Discussion.....	15
2.3.2 IDEs.....	16

2.3.2.1 Pedagogical IDEs.....	16
2.3.2.1.1 Discussion.....	18
2.3.2.2 Professional IDEs.....	19
2.3.2.2.1 Discussion.....	21
2.3.2.3 IDEs w/Command Line Features.....	21
2.3.3 Command Line Environments.....	22
2.3.3.1 Featured Text Editors.....	22
2.3.3.1.1 Discussion.....	24
2.3.3.2 Plain Text Editors.....	24
2.3.4 Summary.....	25
2.4 Programming Environments – Empirical Studies.....	26
2.5 Measures for Programming Environment Evaluation.....	31
2.5.1 Engagement.....	32
2.5.2 Comprehension – Mental Model.....	33
2.5.2.1 Cognitive Learning - Bloom’s Taxonomy.....	34
2.5.2.2 Mental Models.....	35
2.5.2.3 Eye-Tracking.....	35
2.5.2.4 Software Packages.....	36
2.5.3 Comprehension – Understanding Programming Procedures.....	36
2.5.4 Efficiency.....	37
2.5.4.1 Keystroke Level Model.....	37
2.5.4.2 Time on Task.....	39
2.5.5 Usability.....	39

2.6 Summary	41
3. FEATURE SET VARIATION.....	42
3.1 A Continuum of Feature Sets.....	43
3.1.1 Low Assistive Environments	44
3.1.2 Moderately Assistive Environments	45
3.1.3 Highly Assistive Environments	45
3.2 Syntax vs. Drag and Drop Programming.....	47
3.3 Familiarity – Consistency and Affordance of Feature Sets	51
3.4 Summary.....	57
4. ALICEVILLE OUTREACH.....	58
4.1 Demographics	61
4.2 Usability	62
4.2.1 Ease of Use	62
4.2.2 Reliability.....	63
4.2.3 Frustration.....	63
4.2.4 Future Usage	64
4.2.5 Comfort with Programming Robots.....	65
4.2.6 Interest in Computer Science.....	66
4.3 Self-Efficacy	68
4.4 Pennington’s Model.....	69
4.5 Discussion and Summary.....	71
5. CS1 LABORATORY STUDY – PYTHON PROGRAMMING.....	72
5.1 Environments/Experiment Conditions.....	73

5.2 Demographics	75
5.3 Procedures.....	80
5.4 Results.....	82
5.4.1 Self-Efficacy	82
5.4.2 Time on Task	83
5.4.2.1 Observations while Measuring <i>Time on Task</i>	84
5.4.3 Pennington’s Model.....	85
5.4.3.1 Version 1 vs. Version 2 (all three groups).....	85
5.4.3.2 Group Comparison.....	86
5.4.3.3 Question Comparison (all three groups).....	86
5.5 Environment’s Usability Survey.....	87
5.5.1 Initial Impression about the Environment.....	87
5.5.2 Comfort with Environment.....	88
5.5.3 Confidence with Doing Another Assignment with the Environment.....	88
5.5.4 Like the Environment	88
5.5.5 Easiest Attributes about the Environment.....	89
5.5.6 Hardest Attributes about the Environment	90
5.5.7 Experiences with Other Environments (Besides PREOP).....	90
5.6 Discussion.....	95
5.7 Summary	97
6. CS1 STUDY - PYTHON PROGRAMMING.....	99
6.1 Environments/Experiment Conditions.....	100
6.2 Demographics	102

6.2.1 Demographics: First Survey	102
6.2.2 Demographics: Second Survey	103
6.2.3 Demographics: Third Survey	104
6.2.4 Demographics: Survey Comparison	116
6.3 Self-Efficacy	123
6.3.1 Pre-Assessment	123
6.3.2 Second Assessment	124
6.3.3 Final Assessment	125
6.3.4 Change in Self-Efficacy	127
6.4 Comprehension	128
6.4.1 Pennington’s Model	130
6.4.1.1 Section Comparison	130
6.4.1.2 Environment Comparison	131
6.4.1.3 Version Comparison	131
6.4.1.4 First vs. Second Survey Comparison	132
6.4.2 Protocol Analysis – “Think Aloud” Approach	139
6.4.2.1 Summary of Observations – IDLE	139
6.4.2.2 Summary of Observations – VIM	141
6.4.2.3 Results	144
6.4.3 Program Procedures	146
6.4.3.1 Section Comparison	147
6.4.3.2 Environment Comparison	148
6.4.3.3 Question Comparison	149

6.4.3.4 First vs. Second Survey Comparison	150
6.5 Time on Task	160
6.5.1 Exam 0	160
6.5.2 Exam 1	162
6.5.3 Exam 2	164
6.5.4 Final Exam	167
6.5.5 Trends for Proficiency Ratings	171
6.6 Usability Survey.....	173
6.6.1 First Survey	174
6.6.2 Second Survey	180
6.6.3 Third Survey (<i>After Environment Switch</i>)	189
6.6.4 Usability Survey Comparison	200
6.7 Discussion.....	205
6.8 Summary.....	207
7. THREATS TO VALIDITY	209
8. FUTURE WORK.....	211
9. CONCLUSION	213
BIBLIOGRAPHY.....	217
APPENDIX A: LITERATURE REVIEW – PRIOR STUDIES	232
APPENDIX B: LITERATURE REVIEW – MEASURES USED IN PRIOR STUDIES	235
APPENDIX C: ALICEVILLE OUTREACH SURVEYS.....	237
APPENDIX D: CS1-LABORATORY SURVEYS.....	242
APPENDIX E: CS1 SURVEYS.....	254

APPENDIX F: IRB CERTIFICATE 271

LIST OF TABLES

Table 1: Effect on Novice Programmers (From Empirical Studies).....	29
Table 2: Measures for Environment Evaluation (Empirical Studies).....	31
Table 3: Self-Efficacy Descriptive Data.....	68
Table 4: Number of Answers for Each Question.....	69
Table 5: Correct/Incorrect Percentages.....	69
Table 6: CS 160 Demographics	77
Table 7a: Section-By-Section Demographics.....	78
Table 7b: Section-By-Section Demographics (CONT'D).....	79
Table 8: Self-Efficacy Descriptive Data for CS160 (N = 94).....	82
Table 9: Self-Efficacy Descriptive Data Amongst The Three Sections	82
Table 10: Time On Task Descriptive Data for CS160 (N = 91).....	83
Table 11: Time On Task Descriptive Data Amongst The Three Sections (Time in Minutes).....	84
Table 12: Pennington’s Model Version 1 Vs. Version 2.....	85
Table 13: Program Function (Version 2 Modification)	86
Table 14: CS160 Environment Usability Data	92
Table 15a: Section-By-Section Environment Usability Data	93
Table 15b: Section-By-Section Environment Usability Data (CONT'D)	94
Table 16: CS1 Study Outline	101
Table 17a: CS150 Demographics – First Survey.....	105

Table 17b: CS150 Demographics – First Survey (CONT'D).....	106
Table 18a: Section-By-Section Demographics – First Survey	107
Table 18b: Section-By-Section Demographics – First Survey (CONT'D)	108
Table 18c: Section-By-Section Demographics – First Survey (CONT'D)	109
Table 19: CS150 Demographics – Second Survey	110
Table 20a: Section-By-Section Demographics – Second Survey.....	111
Table 20b: Section-By-Section Demographics – Second Survey (CONT'D).....	112
Table 21: CS150 Demographics – Third Survey.....	113
Table 22a: Section-By-Section Demographics – Third Survey.....	114
Table 22b: Section-By-Section Demographics – Third Survey (CONT'D).....	115
Table 23a: CS150 Demographics – Survey Comparison (Section A).....	119
Table 23b: CS150 Demographics – Survey Comparison (Section B*).....	120
Table 23c: CS150 Demographics – Survey Comparison (Section C - IDLE).....	121
Table 23d: CS150 Demographics – Survey Comparison (Section C - VIM).....	122
Table 24: Pre-Self-Efficacy Descriptive Data (N=120) – All Sections.....	124
Table 25: Second-Self-Efficacy Descriptive Data (N=119) – All Sections	125
Table 26: Final-Self-Efficacy Descriptive Data (N=126**) – All Sections.....	126
Table 27: Changes In Self-Efficacy Descriptive Data– All Sections	127
Table 28: Percentage Of Idle/Vim Users (First And Second Surveys)	129
Table 29: Pennington’s Model: Section Comparison (Version 1 vs. Version 2) – First Survey.....	133
Table 30: Pennington’s Model: Section Comparison (Version 1 vs. Version 2) – Second Survey	134
Table 31: Pennington’s Model: Environment Comparison (Version 1 vs. Version 2) – First Survey.....	135

Table 32: Pennington’s Model: Environment Comparison (Version 1 vs. Version 2) – Second Survey.....	135
Table 33: Pennington’s Model: Version Comparison – First Survey.....	136
Table 34: Pennington’s Model: Version Comparison – Second Survey	136
Table 35: Pennington’s Model: Changes in Understanding Programming Concepts – Section Comparison	137
Table 36: Pennington’s Model: Changes in Understanding Programming Concepts – Environment Comparison.....	138
Table 37: Pennington’s Model: Changes in Understanding Programming Concepts – Version Comparison	138
Table 38: Background Information About Subjects [43]	140
Table 39: Task Completion Results [43]	145
Table 40: Challenges for NOT Completing Assignment [43].....	145
Table 41: Weights for Programming Procedures Survey	147
Table 42: Programming Procedures –Section Comparison (First Survey).....	151
Table 43: Programming Procedures –Section Comparison (Second Survey)	152
Table 44: Programming Procedures –Environment Comparison (First Survey).....	153
Table 45: Programming Procedures –Environment Comparison (Second Survey)	154
Table 46: Programming Procedures – Question Comparison (First Survey)	155
Table 47: Programming Procedures – Question Comparison (Second Survey).....	156
Table 48: Changes in Understanding Programming Procedures –Section Comparison.....	157
Table 49: Changes in Understanding Programming Procedures –Environment Comparison....	158
Table 50: Changes in Understanding Programming Procedures – Question Comparison	159
Table 51: Proficiency Rating Descriptive Data Amongst the Three Sections.....	161
Table 52: Proficiency Rating Descriptive Data Amongst the Three Sections.....	162
Table 53: Percentage of IDLE/VIM Users – Exam 1	163

Table 54: Proficiency Rating Descriptive Data Amongst the Environments	164
Table 55: Proficiency Rating Descriptive Data Amongst the Three Sections.....	165
Table 56: Percentage of IDLE/VIM Users – Exam 2.....	166
Table 57: Proficiency Rating Descriptive Data Amongst the Environments	167
Table 58: Proficiency Rating Descriptive Data Amongst the Three Sections.....	168
Table 59: Percentage of IDLE/VIM Users – Final Exam.....	169
Table 60: Proficiency Rating Descriptive Data Amongst the Environments	170
Table 61: CS150 Environment Usability Data – First Survey.....	175
Table 62a: Section-By-Section Environment Usability Data – First Survey	176
Table 62b: Section-By-Section Environment Usability Data – First Survey (CONT'D)	177
Table 62c: Section-By-Section Environment Usability Data– First Survey (CONT'D).....	178
Table 62d: Section-By-Section Environment Usability Data – First Survey (CONT'D)	179
Table 63a: CS150 Environment Usability Data – Second Survey	181
Table 63b: CS150 Environment Usability Data – Second Survey (CONT'D)	182
Table 64a: Section-By-Section Environment Usability Data – Second Survey	183
Table 64b: Section-By-Section Environment Usability Data – Second Survey (CONT'D)	184
Table 64c: Section-By-Section Environment Usability Data– Second Survey (CONT'D)	185
Table 64d: Section-By-Section Environment Usability Data– Second Survey (CONT'D).....	186
Table 64e: Section-By-Section Environment Usability Data– Second Survey (CONT'D)	187
Table 64f: Section-By-Section Environment Usability Data– Second Survey (CONT'D).....	188
Table 65a: CS150 Environment Usability Data – Third Survey (<i>After Environment Switch</i>) ...	192
Table 65b: CS150 Environment Usability Data – Third Survey (<i>After Environment Switch</i>) (CONT'D)	193
Table 66a: Section-By-Section Environment Usability Data – Third Survey (<i>After Environment Switch</i>)	194

Table 66b: Section-By-Section Environment Usability Data – Third Survey (<i>After Environment Switch</i>) (CONT'D)	195
Table 66c: Section-By-Section Environment Usability Data – Third Survey (<i>After Environment Switch</i>) (CONT'D)	196
Table 66d: Section-By-Section Environment Usability Data – Third Survey (<i>After Environment Switch</i>) (CONT'D)	197
Table 66e: Section-By-Section Environment Usability Data – Third Survey (<i>After Environment Switch</i>) (CONT'D)	198
Table 66f: Section-By-Section Environment Usability Data – Third Survey (<i>After Environment Switch</i>) (CONT'D)	199
Table 67a: CS150 Demographics – Survey Comparison (Section A).....	202
Table 67b: CS150 Demographics – Survey Comparison (Section B).....	203
Table 67c: CS150 Demographics – Survey Comparison (Section C).....	204
Table 68a: Individual Evaluation.....	233
Table 68b: Comparison Evaluation	234
Table 69: Related Studies & Sources that Employed Measures of Engagement, Comprehension, Efficiency, and/or Usability	236

LIST OF FIGURES

Figure 1: Programming Environments: Feature Sets [42]	43
Figure 2: Outline of Syntax Programming	48
Figure 3: Outline of Drag and Drop Programming.....	48
Figure 4: Outline of Command Line Programming.....	50
Figure 5: Outline of IDE Programming	50
Figure 6: Advanced IDEs: Moderately Assistive [43].....	51
Figure 7: Pedagogical IDEs/ Graphical Environments: Highly Assistive [43]	52
Figure 8: Microsoft Visual Studio 2008	53
Figure 9: Microsoft Word 2003	53
Figure 10: Command Terminal for Windows7 [43].....	54
Figure 11: Linux Command Terminal - Ubuntu 10.10 [43].....	55
Figure 12: JEdit Text Editor [43].....	55
Figure 13: Notepad (editing window) [43].....	55
Figure 14a: Alice version 2.0.7.....	59
Figure 14b: I-Robot Create	59
Figure 15: Gender Representation	61
Figure 16: Student Classification.....	61
Figure 17: PREOP - Ease of Use	62
Figure 18: PREOP - Reliability	63

Figure 19: PREOP - Frustration.....	64
Figure 20: PREOP - Future Usage.....	65
Figure 21a: PREOP – Comfort with Programming Robots.....	66
Figure 21b: PREOP – Comfort with Programming Robots (Factors).....	66
Figure 22: PREOP – Interest in Computer Science.....	67
Figure 23: IDLE version 2.6.6.....	74
Figure 24: PyScripter version 1.9.9.6.....	74
Figure 25: Notepad/Command Prompt.....	74
Figure 26: CS160 – Time on Task.....	83
Figure 27: IDLE version 3.2 – Linux platform.....	100
Figure 28: VIM version 7.3.35.....	100
Figure 29: Exam 0 – Proficiency Rating (Section Comparison).....	161
Figure 30: Exam 1 – Proficiency Rating (Section Comparison).....	162
Figure 31: Exam 1 – Proficiency Rating (Environment Comparison).....	164
Figure 32: Exam 2 – Proficiency Rating (Section Comparison).....	165
Figure 33: Exam 2 – Proficiency Rating (Environment Comparison).....	167
Figure 34: Final Exam – Proficiency Rating (Section Comparison).....	168
Figure 35: Final Exam – Proficiency Rating (Environment Comparison).....	170
Figure36a. Proficiency ratings – Section A.....	171
Figure36b. Proficiency ratings – Section B.....	171
Figure36c. Proficiency ratings – Section C.....	171
Figure37a. Proficiency ratings – IDLE.....	172
Figure37b. Proficiency ratings – VIM.....	172

Figure37c. Proficiency ratings – UNKNOWN.....	172
Figure37d. Proficiency ratings – BOTH.....	172

1. INTRODUCTION

Programming is a challenge. However, it is a skill that must be developed as a computer science major. Teaching novices how to program has been a challenge of its own. One problem is that students can encounter programming as a barrier and in many cases leave the field of Computer Science. It has been argued that their mental models are not up to par for programming which is a possible reason for this retention problem [35, 163].

Much attention has been applied to improving the retention of incoming CS majors. One focus is the appropriate programming paradigms to teach. Another is the complexity or simplicity of certain programming languages and their effect on the novice understanding.

This attention has also been placed on programming environments. Kelleher and Pausch note that programming environments (and languages) have been built since the 1960s with the purpose of making programming accessible to people of various ages and backgrounds [85]. However, there has been a shift in the focus of environments being used over the years to teach programming. Visual environments like IDEs and robots have become more common for teaching programming. One motivation is to reduce the attrition rate of students majoring in computer science [8]. In addition, visual environments have been used to expose middle and high school students to programming [84, 94, 97]. The objective in many cases was to help students make an easier transition into programming by exposing them at earlier learning stages.

Command line environments are also used to introduce programming to novices, in particular in introductory CS courses. Familiarity, personal beliefs, and anecdotally based motivations of acquiring useful mental models are common reasons for exposing novices to

programming through a command line environment. The appeal to mental models is supported by Chen and Marx who moved their students from an IDE to command line programming [26].

Generally, programming environments vary in features. The level of assistance that these features provide to a novice is important; for example, syntax highlighting, auto completion, or drag and drop coding. Environments with highly assistive features can restrict novices to learn only foundational programming concepts. Tools like Alice and Scratch, for example, can be considered highly assistive. Their drag and drop functionality confines users to learning iteration, sequence, selection, variables, and functions. These environments also restrict exposure to syntax programming. Despite their level of assistance, such environments may not be ideal for a CS curriculum. Environments that have a lower level of feature assistance, such as command line environments, can have the opposite effect. Such environments typically supply the user with the essentials for programming, allowing the user more flexibility while programming. These environments are often used to teach programming at the intermediate and advanced stages of a CS curriculum. Environments that have a moderate level of feature assistance can possess many assistive features, but seldom is the user restricted to learning certain abilities. IDEs and feature rich editors can be considered moderately assistive.

The objective of this research is to determine whether certain tools are potentially more appropriate for teaching novices. One approach is to study the different levels of feature assistance in programming environments. Next, apply appropriate measures to evaluate and compare environments with varying levels of feature assistance.

1.1 Contributions

In this research, novel contributions to address this question are investigated. The structural make-up of programming environments and their behavior can vary. Contributions of addressing the appropriateness of programming environments for novice programmers include:

- **A continuum for explaining the variation and quantity of an environment's feature sets and their effect on how the novice understands programming.** The structural make-up of programming environments varies. Certain features can provide a novice with either flexibility or constraints when learning to program. These features can also influence the learning curve of a programming environment.
- **A tendency for novices to struggle with using a low assistive environment during initial stage of learning this tool.** From the CS1-Laboratory study, it was found that students struggled with using a less assistive environment (Notepad) regardless of their experience with programming. Students were able to use moderately assistive environments (IDLE and PyScripter) more effectively. During the CS1-Study, students who switched to using VIM (low assistive environment) struggled with using this environment. This was particularly true during the protocol analysis and final usability assessment.
- **A moderately assistive environment potentially providing novices with a lower learning curve, while a low assistive environment appears to provide a better understanding of programming procedures.** IDLE students (CS1-Laboratory Study) were able to learn enough about this tool in order to complete the required task despite those who lacked prior programming experience. During the CS1 – Study, IDLE also appeared to have

a lower learning curve during the protocol analysis and final usability assessment. In addition, VIM appeared to have potentially equipped the students with a more helpful mental model for understanding the underlying factors of programming, while enabling them to make easier transitions into using other environments. This was found to be true during the protocol analysis, first programming procedures assessment (*understanding compilation*), and final usability assessment.

- **Systematic methodologies for comparing visual and command line environments and their effect on novices.** Related studies have evaluated the effect of different programming environments on novices. However, majority of these studies only evaluated visual environments (IDEs, robots, drag and drop environments, etc.). Two studies (CS1-Laboratory and CS1-Study) were conducted for evaluating visual and command line environments and their effect on novice programming.

These contributions can serve as preliminary evidence while moving forward with further evaluations of programming environments and their effect on novice programming.

1.2 Research Hypotheses

The objective of this research is to measure the effect of programming environments with varying feature sets on novices to determine if one is more appropriate for learning to program. To answer this question, a set of programming environments will be evaluated through measures of engagement, comprehension, efficiency, and usability (*see Section 2.5 for details*). Due to the constraints that are seen in highly assistive environments, this research will focus on moderately and low assistive environments. The plan is to evaluate both kinds of environments and determine if one is potentially more appropriate for teaching novices how to program. The null hypothesis, H_0 , alternative hypothesis, H_a , and the sub-hypotheses, Ha_{1-4} , are as follows:

H_0 : *A moderately assistive environments is NOT more effective for teaching novices how to program than a low assistive environment.*

H_a : *A moderately assistive environment is more effective for teaching novices how to program than a low assistive environment.*

Ha_1 : *A moderately assistive environment is more engaging.*

Ha_2 : *A moderately assistive environment help novices better understand the concepts and procedures of programming.*

Ha_3 : *A moderately assistive environment is more efficient.*

Ha_4 : *A moderately assistive environment has better usability.*

These sub-hypotheses will support either rejecting or not rejecting the alternative hypothesis that moderately assistive environments are more appropriate for teaching novices how to program. Each sub-hypothesis will be measured empirically.

1.3 Manuscript Outline

Chapter 2 presents a literature review and background discussion about this research. Chapter 3 provides a continuum that compares the varying feature sets within programming environments; in addition to a comparison between visual and command line environments and how novices may interpret the two. Chapter 4 details a pilot study that measured the effect of a particular programming environment on a group of high school students with no prior programming experience. Chapter 5 discusses a study conducted on a CS1 laboratory course involving three programming environments with varying levels feature assistance. Initial data collected from this study along with the results are given in detail. Chapter 6 provides a semester-long study conducted on a CS1 course involving two programming environments with varying levels of feature assistance. Data collected from this study along with the results are given in detail. Potential threats to validity for this research are presented in Chapter 7. Future work and conclusion from this research are provided in Chapters 8 and 9 respectively. This manuscript also provides a bibliography and appendix that list tables, surveys, and other material relevant to this research.

2. LITERATURE REVIEW & BACKGROUND

This chapter provides a literature review and a background detailing the importance of this research. The first three sections discuss previous work for studying novices and their ability to program. Two focal areas in this discussion are appropriate paradigms and languages. A third area is programming environments and their structural makeup. Detailed discussion is presented for each area. Prior studies involving programming environments and their effect on novice programming are presented in the fourth section. The final section looks at appropriate measures for evaluating programming environments and their effect on novices.

2.1 Paradigms

Even though programming environments are the primary focus of this research, how a student is taught to program can be a factor. A critical concern is whether CS majors can be turned into expert programmers after four years of a CS curriculum [165]. However, there remains the issue of whether novices can matriculate through a CS curriculum and acquire the necessary skills to help them become successful programmers. Certain programming paradigms or concepts may be too complex for a novice to understand. Their complexity could potentially play a role in the “barrier” that many novices face while learning to program. The next two sections discuss debates about appropriate paradigms for teaching introductory programming.

2.1.1 OO vs. Non-OO Programming

There have been discussions about how introductory courses should expose novices to programming. One discussion concentrates on the idea of exposing novices to object-oriented programming at the introductory level. Researchers have termed this approach as “objects first” [4, 22, 33, 80, 134, 155]. However, this approach has been opposed by others because it may be inappropriate for novices to learn at such an early stage [5, 40, 133]. Other approaches have considered functional programming (ex. Miranda and Scheme) [21, 64, 124, 164], procedural programming (ex. Ada, Pascal, and C) [9, 51, 137], script programming (ex. Python) [148], and reflective programming (ex. Logo) [146].

2.1.2 Concepts in a CS1 Course

Appropriate programming concepts that should be taught in a CS1 course have also been disputed. Schulte [144] conducted a study that polled concepts usually taught in a CS1 course. An online survey was sent to 477 CS instructors. He received 349 responses. From these responses, Schulte found 28 different topics to be appropriate for teaching an introductory CS course. These topics represent both OO and non-OO programming. Out of these topics, he saw that the most common concepts taught in CS1 courses were: *Selection&Iteration*, *Simple Data Structures*, *Parameters*, *Scope*, *Objects&Class*, and *Syntax*. The most difficult concepts to teach were: *Recursion*, *Algorithm Efficiency*, *Polymorphism& Inheritance*, *Generics*, and *Advance Data Structures* [144].

Schulte's study was preceded by an earlier study conducted by Dale at the University of Texas [38]. Dale issued an online survey to several CS instructors with 351 respondents. She found that the more emphasized concepts (based on non-OO programming) were: *Information Hiding*, *Selection*, and *Repetition*; for OO-programming only *Code Reuse* was commonly emphasized in a CS1 course [38].

Some of the concepts shown from Schulte and Dale's respective studies are considered in this research; in particular selection, iteration, information hiding, scope, repetition, syntax, and semantics. The studies discussed in Chapters 4-6 explored some of these concepts and others that are common in a CS1 course.

2.2 Languages

The easiness or complexity of certain languages for an introductory sequence has also been discussed. Schneider [143] believes that a programming language should have richness and simplicity – rich enough to introduce the necessary fundamental concepts in programming but simple enough for a novice to grasp in a one semester course. Others argue that certain object-oriented languages do not meet these ideal standards. For example, Clark et.al [27] believe that Java is great to teach as the first language due to its educational benefits as well as commercial behavior, but later argue that simple programs such as *HelloWorld.java* can come off as intimidating to students. Others dispute that the structure of object-oriented languages (like Java and C++) can be confusing, frustrating, intimidating, or just simply inappropriate for first-year CS majors [9, 15, 27, 28, 61, 104, 119]. However, there has been work done in attempt to lessen the complexity of object-oriented languages like Java. For example, Roberts developed MiniJava, which is a simplified version of Java [138].

Python has been discussed in terms of being an appropriate language for a CS1 course [16, 44, 127, 132, 148]. It has been argued that this language is easy to learn, extensible in complicated languages like C or C++, and powerful enough to support features that are typically seen in object-oriented languages. In addition, Python has been argued to be an object-oriented language even though it is occasionally seen as a scripting language [57]. Researchers have also argued that languages like Python, Logo, Eiffel, Scheme, and Pascal, have been developed with the intent of being used in introductory level courses [9, 21, 64, 119, 146].

There are other languages that could be used in introductory programming courses, but have been discarded due to their lack of popularity. Becker [9] mentioned that Pascal is considered ancient, not commercially marketable, a step backwards, and unpopular because it does not possess C-like syntax. Ada was considered great to use in the 1980's because of its "pure view of modularization, object-oriented programming, and good software-engineering methodology;" however it is selectively used by certain schools throughout today [28].

2.3 Programming Environments - Overview

Programming environments provide programmers with the necessary tools for developing and implementing their code. These environments come in different formats and styles. Some environments are built specifically for a particular language (ex. BlueJ, DrJava, and DrScheme) while others can support multiple languages (ex. Eclipse and Microsoft Visual Studio). There are also environments used for advanced and professional programming [49, 68, 75, 76, 102, 110, 115], while others are utilized for novice programming [2, 19, 45, 46, 78, 82, 91, 95].

Kelleher and Pausch's taxonomy categorizes programming environments based on social learning, motivation (easing intimidation), code understanding, language understanding, entertainment, and education [85]. These environments can also be categorized as specialized applications, integrated development environments, or command line environments. Integrated development environments can be further categorized into *pedagogical* and *professional*. The remainder of this section will go into a detailed discussion about each category.

2.3.1 Specialized Applications

Specialized applications were created to conduct programming on specific tasks. These applications possess features such as drag and drop coding, virtual worlds, robots, graphic visualization, and animation. Some of these applications have been used to help novices understand fundamental logic behind programming. For example, *Alice* is an environment that introduces programming through a 3D virtual world. It was developed at Carnegie Mellon

University with the intent to help novices understand programming techniques through an object-oriented environment. In Alice, programmers are exposed to objects and classes with a more hands-on approach by using drag-and-drop procedures to develop programs. A main goal is to help programmers make a successful transition to languages like Java and C++ where classes and objects are more common [95]. Another attribute is that the programmer can interact with the environment from inside an Emacs editor [30], which can be considered a command line editor on the Linux platform. However, Emacs has also been classified as an IDE because of its features [29].

Karel the Robot has been used in CS1 to teach “fundamental concepts and programming skills – quickly and easily” [118]. Karel’s background is based on a world that contains avenues that run in the north and south direction numbered one to infinity and streets that run in the east and west direction that are numbered in the same manner. Objects such as walls, beepers, and various robots can exist in this world. According to Becker, advantages of using Karel the Robot include [9]:

- object-oriented programming concepts
- visual representation that provides animated feedback to the programmer
- student engagement/enjoyment with using robots to do many tasks
- visual output being more amusing than textual output.

A related application involving robots is *LEGO Mindstorms*. Lawhead et al. [89], talks about the use of a *LEGO Mindstorms Robotics Invention System* to teach object-oriented programming at the introductory course level. Java is a common language for this environment [6, 89]. The robot itself can be built via possessing a Lego Mindstorm kit, programmable brick (RCX), active sensors, and motors [7, 87, 116]. The brick can be programmed using LEGO software that provides the option of either a GUI or command line to test programs. The LEGO

approach is believed to have some of the same advantages as seen from Karel the Robot along with other benefits such as [89, 99]:

- exposure to programming at different levels
- appealing to women and minorities
- appealing to students with intrinsic motivation for learning to program.

As an extension to LEGO, there is a software interface called *LabVIEW*. This software is able to manipulate LEGO motors and lights and interpret behavior from the LEGO's sensors by being connected to the serial port of a computer [50]. This software can empower a library of subroutines and virtual instruments which make up the component of RoboLab. Similar to features seen in IDEs and related visual environments, LabVIEW allows users to perform interaction via mouse clicks.

Kelleher and Pausch discussed a robotic environment called *Robocode* [85]. Created by Matthew Nelson [109], this environment exposes novices to Java by programming a robotic battletank. Nelson believes that this style of learning should give a novice enough motivation to overcome the intimidation and hurdles of programming [85, 109].

There are also applications available for helping novices understand algorithmic behavior. For example, *Raptor* provides visual a representation for students to picture the behavior of their written algorithm. Raptor was developed at the United States Air Force Academy with the intent of using graphics to teach students how to develop and understand algorithms. In addition, the developers wanted to address the issue of poor syntax usage by novices which was common when using non-visual environments [25]. Raptor is derived from the phrase Rapid Algorithmic Prototyping Tool for Ordered Reasoning. The general purpose for Raptor is to visually enable students to see the execution of their algorithm through a step-by step process. This environment

can also show the location of an executed object, symbol, class, etc, at a particular moment during this process [25].

Similar to Raptor, *Jeliot* is a tool that provides an active view of a written program [93]. Jeliot was developed at the University of Helsinki under the direction of Jorma Tarhio and Erkki Sutinen [12, 13]. According to Ben-Ari, who led in the creation of Jeliot 3, this tool was built with the intent that students should be “learning by doing” and have the necessary tools that will help them “construct a visual representation of a program” [12]. While using the Java language, Jeliot’s window displays the source code of a program in the left pane. The animation in relation to the source code is displayed in the right pane. Button widgets are also located in the lower region of the window to control the animation (Note: sources [13, 93, 104] provides snapshots of either Jeliot 2000 (the original Jeliot tool) or Jeliot 3). The researchers evaluated Jeliot in a high school setting that involved sophomores who were taking a year-long CS course. They found that the Jeliot application improved the students’ knowledge and understanding of the assigned projects because of the animation provided [93]. Even though their subjects were high school students, this tool is believed to produce similar results at the college level. Ben-Ari believed that a student’s attention-span would also improve because their engagement with this visual tool [12, 13].

2.3.1.1 Discussion

Specialized applications are typically created for conducting specific tasks in programming. These applications possess animation and other visual attributes that have been used as mechanisms for teaching novices how to program. According to researchers, who studied these tools, some of these applications can be effective for teaching novices how to program.

2.3.2 IDEs

Integrated development environments (IDEs) represent another set of tools that are used for programming. A typical IDE contains a text editor and a built-in compiler or interpreter. More advanced IDEs may have a built-in debugger [114]. Many of these environments possess features that assist the user while programming. To compile or execute a written program, the user may click on a “compile” or “run” button as an alternative to a command line argument [135]. Based on their structural make-up and feature sets, IDEs can be considered either *pedagogical* or *professional*.

2.3.2.1 Pedagogical IDEs

Pedagogical IDEs are built specifically for novice programmers. These environments are simple because they lack many features with the intent to appeal to novice programmers. They are considered “half-strength” because they are structured to handle smaller projects [154]. These environments generally attempt to reduce the learning curve as well as to ease underlying complexities of programming for inexperienced programmers.

For example, Peter DePasquale III built an environment called *CSI Sandbox* at Virginia Tech. The student’s interface for this environment consist of one button, three menus, a text editor for entering code, list of errors made, and display of the cursor location. It was DePasquale’s intention to design this environment to not be cluttered with many buttons, menus, and text areas that are more prevalent in commercial IDEs [41].

DrJava is another IDE with similar traits. This environment was developed at Rice University with the intent of eliminating the “intimidation factor” seen by novice programmers when faced with the challenge of writing Java code [3]. *DrJava*’s interface is built with an

incremental behavior known as read-eval-print loop (REPL) which is geared toward helping a novice with program development. Its window consists of a two paned window: *interactions* and *definitions* pane. The interactions pane enables a user to type in Java expressions and/or statements while the results are displayed immediately. The definition pane enables a user to write code that may be a little more sophisticated such as class definitions. This particular pane also provides feature assistance such as syntax highlighting, brace matching, and automatic indenting [3]. The general interface is designed to be less distracting due to its simple layout while being reasonably manageable to novices [135].

BlueJ, a similar Java IDE, was developed at Monash University. Similar to DrJava, its purpose was to improve how novices interact with an environment while programming. However, the interaction in this environment may be different from other pedagogical environments. Kolling made an indication that BlueJ represents a graphical interaction for users while other related systems focus more on the lines of codes [88]. Another difference is the ability to expose students to UML diagrams.

Olan [124] compared BlueJ to DrJava through evaluating their simplistic behaviors. He found that BlueJ is richer in features but mentions that this attribute may give DrJava an advantage. Another observation was that DrJava's one window interface may be more fitting for novices when compared to BlueJ. Even though BlueJ's interface is more graphical, its functionality may lead to more windows that a novice will have to navigate. A critical observation of Olan was the ease of transition to different programming environments. Olan noted that BlueJ's developers acknowledged the possibility of student having trouble transitioning from this IDE to a more advanced/sophisticated one [88, 114]. In contrast, DrJava's developers believed that their environment could be used beyond introductory level

programming [114]. Allen argued that novice programmers waste time using BlueJ because not only do they have to learn the Java language but also the protocols of using the interface. Allen also indicated that BlueJ does not offer assistive features to novices such as brace matching and syntax highlighting [3].

There are other pedagogical environments that are built with similar reasons as DrJava and BlueJ. *JGrasp*, for example, also utilizes UML diagrams for visualization. This environment was created at Auburn University with the goal of using visualization to improve the “comprehensibility of software” for novice programmers [36]. Another system is *DrScheme* which was developed at Rice University with an interface somewhat related to DrJava such as a definition and interaction pane. DrScheme also has a control panel and a menubar that consists of five buttons. This environment helps novices learn how to program in the Scheme language [53]. *Jeroo* was developed to help students master the fundamentals of object-oriented programming. This tool succeeds a tool called *Jessica* which was created by Lai Kuan Tong in 1990. Jeroo contains its own language but is argued to have similarities to Java and C++, which helps the users make an easier transition to such languages [142].

2.3.2.1.1 Discussion

When looking at the literature concerning pedagogical IDEs, the developers created these tools with a common objective. The objective was to create an environment(s) specifically for novice programmers in order to assist in the necessity of learning to program. However, tools like BlueJ may expose novices to “pitfalls”. Such pitfalls make the reliability of certain pedagogical environments questionable in regards to their overall intent of being appropriate for teaching novices how to program. This also raises a question of whether some of these

environments have a deceptive appearance of a simple IDE but in actuality are no better than some of the IDEs that are considered advanced/sophisticated.

2.3.2.2 Professional IDEs

Another class of IDEs is those with advanced/sophisticated or professional behavior. Professional IDEs are typically rich in features, but their complexity may be too much for a novice programmer to handle. In many cases, novices spend more time learning how to use these tools rather than learning the intended concepts and paradigms of programming [119]. However, it has been argued that these particular IDEs may help novices in the long run because they give exposure to the kinds of environments that users more likely used in the “real world” [26]. Because of their many features, professional IDEs are considered “full-strength” tools [154].

Despite their complexities, professional IDEs do offer programmers quality assistance with writing code. For example, *Eclipse* is considered productive as well as efficient for developing programs in Java. Some of the features seen in Eclipse consist of [26]:

- syntax highlighting for keywords
- code auto-completion for variables and predefined methods
- code assistant that gives method hints
- package importing
- wizards to eliminate manual repeated typing for classes, methods, constructors, etc.
- a package-class hierarchy view and a class fields and methods outline view
- javadoc documents
- user preference on indentation, color, fonts, general project/class comments, etc.

Another feature that Eclipse provides is a plug-in for altering its functional behavior. One study showed Eclipse being altered to behave as a pedagogical environment in order to suit the needs of novices [135, 136]. Eclipse is primarily free to download and widely used in industry [26].

Coleman et al. gave a presentation called *Java IDE Shootout* where Eclipse and other Java IDEs were evaluated. According to Coleman, Eclipse provides excellent plug-in support, syntax highlight, and spellchecking. One of the bad qualities was that Eclipse has a high learning curve. Coleman also indicated that Eclipse was too generic which makes it hard for new programmers to grasp [29]. Other environments included in the discussion were *NetBeans*, *IntelliJ* and *JDeveloper*. Coleman argued that all of these environments had features too complex for novices to use during early stages of programming.

Microsoft Visual Studio is another environment rich in features. According to a written overview of this system, Microsoft Visual Studio enables users to develop Smart Client Applications, build Window Vista Applications, and develop web applications [101]. The main idea for this environment is to improve the user's programming experience. However, no direct indication was made about improving the experience of a novice programmer. Because of Microsoft Visual Studio's make-up and rich features, it can be considered a professional IDE. Microsoft Visual Studio was created by Microsoft with the intended purpose of integrating various tools onto a primary user interface, whether software creation, compilation, editing, or development is being conducted [17]. This environment incorporates many languages such as C/C++, Visual Basic, and C#. It is also able to support languages such as XML, HTML, and JavaScript while languages like Ruby, Python, and many others can be installed onto the environment.

There has also been discussion about environments that may be more fitting for programming at the intermediate stage. For example, *Jenuity* has been used by its developers' institution for some time [160]. They found that when comparing Jenuity to other environments like Eclipse, NetBeans, and a BlueJ edition of NetBeans, Jenuity is more efficient based on start-

up time and has better memory usage. However, no direct indication has been made concerning Jenuity's capability of being used at earlier stages of a CS curriculum.

2.3.2.2.1 Discussion

Professional IDEs can possess numerous features for program development. In many cases, these environments may be too complex for a novice to use. However, researchers have argued that these IDEs can be more effective for novices to use in the long run. One reason is their assistive features which include efficient coding, error highlighting, syntax highlighting, and auto brace matching. These and related features can provide novices with "shortcuts" when writing a program. In addition, these environments have plug-in support for altering (and limiting) their feature sets to portray the behavior of a pedagogical IDE.

2.3.2.3 IDEs w/Command Line Features

There is a case where a command usage is one of the features in an IDE. The *Anjuta* IDE, for example, was built for the GNU/Linux platform and supports both C and C++. This environment is versatile because not only does it possess the behavior of a typical IDE such as project management, application wizards, an interactive debugger and syntax highlighting, but also the ability to use command line features [57]. In the case of novice programmers, there is the concern of whether this environment and others like it cause confusion because of this dual (visual and command line) behavior.

2.3.3 Command Line Environments

There are programming environments that utilize a command line terminal as part of their behavior. Many of these environments are better known as *text editors*. These editors require some command line interface or terminal to conduct program compilation, interpretation, or execution. For example, if a programmer is programming on the Windows platform, *Notepad* and editors like *UltraEdit* [67] are available for writing applications and the command prompt terminal is used for compilation, interpretation, and execution of a program. On Linux and Unix operating systems, a command line shell or console is used to retrieve commands from the user in order to perform specific tasks [90]. Example editors that perform such behavior are *Emacs*, *Vi/Vim*, and *Pico*.

Some command line environments possess similar features that are seen in IDEs. There are also editors that hardly possess any features. Such environments are considered to be plain text editors. Command line environments can be categorized into two groups: *featured text editors* and *plain text editors*.

2.3.3.1 Featured Text Editors

Editors in this class can possess relative features that are seen in IDEs. However, these editors may still require command usage and a terminal for manipulating written programs. For example, Coleman [29] argued that *Emacs* is an IDE because of its attributes. He noted specifically that Emacs possesses features of auto indentation, code completion, syntax highlighting and customizable plug-ins. However, Coleman mentioned that Emacs possesses a finite set of commands for programming, while mouse usage is not necessarily needed. In regards to novice programmers, the question remains concerning the effort for a novice to learn

and understand an editor like Emacs. Coleman admitted that one of the challenges for using Emacs is the learning curve for key binding commands [29].

Vi/Vim is another featured text editor. Vi was derived as an abbreviation of the word “visual.” It was first introduced in 1976 by Bill Joy at the University of California-Berkeley as a descendant of *ex* and *ed* which were text editors that only displayed text one line at a time [162]. Vi was the first full-screen text editor on the UNIX platform. VIM, also known as *Vi Improved*, was created in 1991 through the works of Bram Moolenaar. This editor was built to be more configurable as well as more efficient for text editing. VIM possesses all of the functionality of Vi in addition to other features that enhance program production, given its name. It is often called a “programmer’s editor” and has been considered an IDE because of its useful features for programming, which include syntax highlighting and mouse usage. However, Vi/Vim is a tool that must be learned, because it is not designed to cater to the users’ every need [103].

Other editors that are represented in this category are *UltraEdit*, *MultiEdit*, *SlickEdit*, *KEDIT*, *SEDIT*, and *X2*. Even though these editors are not labeled as an IDE, they possess similar features. For example, all of these editors, except X2, allow for user interaction by mouse clicks (not for compiling or executing a program with exception to SlickEdit). The X2 editor is all command based. The editors, UltraEdit, SlickEdit, KEDIT, and X2, provide syntax highlighting for their users. Many of these editors, however, still rely on command arguments and terminals for program manipulation. For example, UltraEdit, MultiEdit requires the user to open a command prompt in order to compile and execute written programs. KEDIT, SEDIT, and X2 are similar to Vi/Vim in Linux where the user has to type a certain command in its editor in order to access the terminal shell for compilation and execution. SlickEdit by far is the closest to resembling an IDE because the user can actually execute a program using the mouse. It has even

been considered a full IDE [26]. In contrast, it has been considered not to be an IDE but rather an editor “on steroids” because of its many features [20].

2.3.3.1.1 Discussion

These particular editors can possess relative assistive features that are seen in visual environments. However, their overall behavior resembles that of a command line environment [20, 69, 83, 106, 152, 153, 157]. In regards to novice programming, there is no direct evidence or studies that show these editors to be beneficial for teaching novices how to program.

2.3.3.2 Plain Text Editors

Many of these editors do not possess features that are seen in IDEs or featured text editors. In many cases, the user is provided with the essentials for programming. For example, the *Notepad* editor on the Windows platform can be used to write programs. However, its interface does not provide feature assistance to a programmer. When using Notepad, the command prompt terminal must be used for program compilation, interpretation and execution.

Editors with similar functionality are *XEDIT*, *THE*, and *Pico*. *XEDIT* is an editor that fully relies on commands for its functionality. In addition, this editor functions through driven command arguments that allow exiting, saving, or loading files into its editing window. It also relies on commands to access its terminal window in order to execute a written program. *THE* (The Hessling Editor) is an editor modeled after *XEDIT*. It is considered an “orthodox” editor due to the fact that it can work on specific types of syntax in a source code based on the given commands [63, 159]. *Pico* is an editor considered to be an alternative to *Vi/Vim* or *Emacs* due to its simplicity [122, 128]. Unlike editors that even require commands for moving the cursor, *Pico*

provides a straightforward functionality, which does not require the user to remember commands [122, 128]. Overall, these environments have a basic functionality. However, there is no direct evidence or studies that show whether these environments can suit the needs of a novice programmer.

2.3.4 Summary

When studying programming environments, there are tools built for a specific task(s) (Specialized Applications), with user-friendly features (IDEs), or to be command driven (Command Line Environments). Specialized applications and pedagogical IDEs have been used to assist novices with programming. However, there has been discussion about whether some of the lightweight IDEs are deceptive to this purpose.

Professional IDEs have many features at the user's exposure. However, the quantity of these features may increase the learning curve for novices. Command line environments may impose a related concern. Even though some text editors have assistive features while others have been declared simple, there is no direct evidence to show that these editors are able to assist novices with programming. The next section details prior studies for measuring programming environments and their effect on novices.

2.4 Programming Environments – Empirical Studies

Previous studies have shown the impact of programming environments on novice programming to be positive, negative, or none (Table 1). Measurements involved in these studies were subjective (ex. attitudes) or objective (ex. time on task). Some environments were evaluated individually while others were directly compared to other programming environments. A detailed summary of each study is provided in Tables 68a and 68b in Appendix A. Many of these studies were conducted to measure the impact of visual environments due to their inclusion into introductory sequences over the last decade. However, research regarding command line environments and their effect on novice programmers is not as extensive.

Hagan and Markham [62] did a study on the effects of BlueJ to teach object-oriented programming in an introductory course. They found that students had a neutral attitude towards BlueJ at the beginning of the semester. They believed that the difficulties of installing and learning to use the system may have influenced these feelings. As the semester went along, however, the students' attitude gradually became more positive towards BlueJ [62].

Moskal, Lurie, and Cooper [32] measured the effect of Alice on novices during a two-year study. One of their main targets was novices who were considered at-risk (*students with little or no programming experience prior to CS1 enrollment or a weak mathematical background*) [32]. The authors concluded that Alice had a constructive impact on performance, retention, and attitudes of novice programmers, especially on students who were considered at-risk [32].

DePasquale [41] evaluated his CS1 Sandbox environment (with and without language subsets) against Microsoft Visual C++.Net. He found that the students learned and performed equally well with either CS1 Sandbox (both versions: language subsets or not) or Microsoft Visual C++.Net. DePasquale did discover that the students who were using CS1 Sandbox at the beginning of the semester later migrated to using Microsoft Visual C++.Net. He also found that the application of language subsets to CS1 Sandbox enabled students to be more efficient with their tasks than those using Microsoft Visual C++ [41].

Chen and Marx [26] performed a study over a period of two years that evaluated the usage of Eclipse against an environment called Ready to Program in a CS2 course. The first experiment took place in the fall of 2003. The students enrolled during this time preferred Eclipse over Ready to Program because of their excitement when introduced to the environment during an in-class demonstration. However, most of these students chose Ready to Program to do their take-home projects. Some of the reasons for not using Eclipse were based on lack of experience, issues with downloading the software, and the difficulty of using this environment without the direct guidance from the instructor [26]. Students in the following two semesters (Spring and Summer 2004) showed slightly better attitudes toward Eclipse. The authors did mention that these particular students received a CD that provided hands-on exercises for using Eclipse, which could explain the attitude change. During that following semester (Fall 2004), the authors experimented with both CS1 and CS2 courses by using Eclipse as the programming environment. They found that the students depended too much on the wizards that Eclipse provides with insufficient understanding of what they were doing [26]. Therefore in the Spring 2005 semester, no IDE was used for programming but rather Notepad and the Command Prompt terminal. The reason for the change was to help the students get a broader understanding of

compilation, execution, and editing of programs. The authors also believed that this change would help the students better understand the usefulness of an IDE [26]. Unfortunately, data concerning the effect of Notepad on these students was not collected.

Carlisle, Wilson, Humphries, and Hadfield [25] conducted a three semester study to measure the effect of Raptor, a visual programming environment for teaching algorithmic problem solving, on the students' ability to learn algorithmic problem solving when compared against MATLAB [25]. Students were required to implement three algorithmic designs (Enumeration, Bowling, and SARS) using their respective environment. They found that students using Raptor performed significantly better when implementing the Enumeration and SARS design. For the Bowling design, the student using Raptor did significantly worse than the students using MATLAB. The authors believed the outcome for the Bowling design was influenced by the challenging usage of arrays in the Raptor environment. A survey was only given to the students in the latter two semesters to measure the ease of use for Raptor; it consisted of ten questions on a 7-point Likert scale. The students' ratings were above the neutral rating for 9 of the 10 questions. The one question that had a lower than neutral rating focused on how much the students enjoyed programming. The authors believed that this result was due to the idea that programming was the students' least favorite thing to do in the course [25].

McWhorter and O'Connor [99] performed a study to determine if LEGO® Mindstorms influenced motivation for students taking a CS1 course. They found that the group of students using LEGO Mindstorms showed a significant decrease in extrinsic motivation from the control group. They concluded that LEGO Mindstorms hardly had any considerable effect on the students' motivation for programming [99].

Table 1: Effect on Novice Programmers (from Empirical Studies); * represents anecdotal evidence

Environment	Visual or Command Line	Positive Effect	Negative Effect/No Effect	Specific Effect
BlueJ	Visual	Yes - <i>Gradual</i>	No	Attitudes
Alice	Visual	Yes	No	Performance, retention rate, and attitudes
CS1 Sandbox	Visual	Yes – <i>with subsets</i>	No	Time on task
Eclipse	Visual	No	Yes	Complexity of usage
Raptor	Visual	Yes	No	Performance, and ease of use
LEGO® Mindstorms	Visual	No	Yes	Extrinsic Motivation
Notepad*	Command Line	Yes	No	Broader understanding of programming fundamentals

These studies showed conclusions about the impact of programming environments on novices. Some of these studies found certain environments to have a positive impact. For example, BlueJ influenced a positive attitude from the students based on the study performed by Hagan and Markham. It was also found that these students were able to grasp the concepts of object-oriented programming much easier. In Moskal, Lurie, and Cooper’s study, Alice not only had a positive influence on the students’ attitude but there was also a significant improvement in their final grades and the retention rate. DePasquale discovered that students are potentially more efficient with their task when language subsets are applied to the CS1 Sandbox. Carlisle, Wilson, Humphries, and Hadfield learned that the Raptor IDE caused a significant increase in the students’ correctness for learning how to perform algorithmic problem solving. They also concluded that these students found Raptor easy to use.

In other cases, there were studies that showed the potential of programming environments having either a negative or no impact at all on beginning CS majors. For example, Chen and Marx found that the appearance of Eclipse excited their students, but its complexity caused the authors to move later students to command line programming. McWhorter and O'Connor found that LEGO Mindstorms had an insignificant effect on their students when compared to the traditional approach but also caused a negative effect on the students' extrinsic motivation. These studies measured the effect of certain programming environments on beginning CS majors. With the exception of Chen and Marx's study, these studies did not mention any description, evaluation, or comparison of command line environments.

2.5 Measures for Programming Environment Evaluation

Guzdial advocates that “the greatest contributions to be made in this field are not in building yet more novice programming environments but in figuring out how to study the ones we have” [60]. Measures used for environment evaluation have varied. These measures have been either quantitative (such as efficiency, error rates, retention rates, or grades) or qualitative (such as ease of use, feelings/attitudes, or motivation/self-efficacy). The empirical studies in the previous section employed some of these measures for evaluating the respective environments (Table 2). Table 69 lists related studies and sources that have either researched or employed these measures (Appendix B). This section discusses appropriate measures for evaluation and their respective categorization: engagement, efficiency, comprehension, and usability. Each category is discussed in detail in the following sections.

Table 2: Measures for Environment Evaluation (Empirical Studies)

Environment	Specific Effect	Measure Categories
BlueJ	Attitudes	Engagement
Alice	Performance, retention rate, and attitudes	Efficiency/Comprehension/Engagement
CS1 Sandbox	Time on task	Efficiency
Eclipse	Complexity of usage	Usability
Raptor	Performance, and ease of use	Efficiency/Usability
LEGO® Mindstorms	Extrinsic Motivation	Engagement
Notepad*	Broader understanding of programming fundamentals	Comprehension

2.5.1 Engagement

This measure determines whether programming environments provide some level of comfort, motivation, or attraction for students learning to program. There are two studies that measured potential factors for success in introductory programming courses. In one study, Wilson and Shrock [165] measured twelve factors that may influence how well novice programmers would do in early programming courses. Out of these factors, a programmer's *comfort level* had the highest influence on success. The other study was conducted by Bergin and Reilly [14] who measured the effect of *motivation* (intrinsic, *ex. self-enjoyment*; extrinsic, *ex. money*) or *self-efficacy* on success with programming. They found that students who were intrinsically motivated tend to perform better than those who were extrinsically motivated, which also supported previous work done by Lumsden [96]. In Bergin and Reilly's study, they used tests like the Rosenberg Self-Esteem Questionnaire [140], Computer Programming Self-Efficacy Scale [129, 130], and Motivated Strategies for Learning Questionnaire (MSLQ) [123] for measurement.

The Rosenberg Self-Esteem Questionnaire was created by Morris Rosenberg in 1965 as a method for measuring the level of a person's self-esteem. This test is composed of ten questions using a 4-point Likert scale. Each question is scored based on the provided response. For some questions, strongly agree is scored with the highest amount, which is 3, while strongly disagree is given a score of 0; other questions are scored vice versa. The scale of this questionnaire ranges from 0-30, where the score of 30 indicates that a person has high self-esteem [140].

There are other tests available for measuring related factors of engagement [58, 107, 158]. In particular, Ramalingam and Wiedenbeck developed a Computer Programming Self-Efficacy Scale [129, 130]. Unlike other tests that primarily focus on the general usage of a

computer, Ramalingam and Wiedenbeck's test concentrates on how a student feels about their programming performance. This test consists of 32 questions that fall into one of the four following factors: *independence and persistence*, *complex programming tasks*, *self-regulation*, and *simple programming tasks*. Each question is based on a 7-point Likert scale, where 1 indicates *not confident at all* and 7 indicates *absolutely confident* [129]. The authors performed a pretest and a posttest using this scale on 421 students who were enrolled in one of eight sections of an introductory computer science course. More information concerning the actual results can be found in their paper [129].

The Motivated Strategies for Learning Questionnaire [123] was created by a group of researchers at the University of Michigan. This tool is used to measure a student's motivation and strategies for learning. The MSLQ is broken into two sections (Motivation and Learning Strategies) that consist of thirty-one questions each. Each question has a 7-point Likert scale. McWhorter and O'Connor used MSLQ to measure whether LEGO Mindstorms motivated students in CS 1 [99].

2.5.2 Comprehension – Mental Model

Incoming CS majors tend to struggle with learning the fundamentals of programming and problem solving techniques. One reason is due to the difficulty of comprehending a program. Program comprehension is important for reasons such as: it allows programmers to be more effective at completing other tasks in a program [164], it is very helpful for novices to be successful at debugging a program [59, 77, 108], and it is crucial for them to be able to extract necessary information from a snippet of code in order to make any intended modifications to a program [65].

One observation made by Adelson [1] is that novices tend to have a different mental model from expert programmers when dealing with code. She concluded that novices tend to develop a mental representation that consists of concrete information, while experts develop a mental model consisting of functional information [1]. Winslow adds that novice programmers tend to lack detailed mental models when thinking through a program [166]. Wiedenbeck and Fix listed five abstract characteristics of an expert's mental representation [164]: being hierarchical and multi-layered in their thinking process, showing explicit mappings between different layers, recognizing basic programming patterns, being well connected internally, and being well grounded in the program text. Wiedenbeck and Fix conducted a study utilizing these characteristics to verify the difference in behavior between novices and experts. The results of this study are detailed in their paper [164].

There are measurements available for measuring a programmer's ability to comprehend written programs. These measurements can be applied in the form of cognitive learning, mental models, or eye-tracking. In addition, there are software packages available for conducting this measure. The following sections discuss each measurement in further detail.

2.5.2.1 Cognitive Learning - Bloom's Taxonomy

Cognitive learning primarily looks at a person's mental behavior. Reasoning, perception, and understanding play respective roles in this behavior. Bloom's Taxonomy is a primary model for measuring cognitive learning [86, 98, 108]. This taxonomy looks at the cognitive process of a human from six different levels: *knowledge*, *understanding*, *application*, *analysis*, *synthesis*, and *evaluation* [18]. In the case of novice programming, employing Bloom's Taxonomy would

measure how students reason, perceive, and understand the syntax, semantics, and behavior of a program.

2.5.2.2 Mental Models

In relation to cognitive learning, a person's thought process can represent his/her mental model. In regards to program comprehension, there are many mental models available for measurement [31, 34, 39, 54, 56, 66, 92, 113, 120, 131, 147, 151, 153, 161]. However, Pennington's model has been used most frequently and "is the basis for much subsequent research in this area" [86]; it is even the parent model to some of the later models developed [56, 66, 113, 131, 147]. Kelly provided a table that consisted of an overview of ten comprehension models; one column in particular contained the number of levels from Bloom's Taxonomy used by each. Pennington's model possessed the most levels with three [86].

Pennington's model, in particular, proposes five characteristics for measurement: elementary operations, control flow, data flow, program function, and program state [120]. These characteristics also provide a good representation of the kinds of concepts that are typically taught in a CS1 course. This model has been used for measuring program comprehension in other studies [130, 131, 163]. Ramalingam and Wiedenbeck, in particular, used this model to compare object-oriented programming against procedural programming [131, 163].

2.5.2.3 Eye-Tracking

Eye-tracking is considered valuable because it can potentially provide detailed observations of a programmer's behavior [37]. Duchowski has written articles and books on the theory and practices of eye-tracking [46, 47, 48]. Many measurements have also been used in

regards to eye-tracking. For example, there are studies that focus on the web as a target of observation [37, 79]. Another study uses eye-tracking to measure comprehension for UML class diagrams [168]. There are also studies that directly focus on users' comprehension of written algorithms [10, 11, 35, 111]. In addition, some of these studies used physical devices such as: a restricted focus viewer (RFV) [74, 121, 139], a remote eye-tracker [10, 11], and trackers design for laboratory usage [70, 100, 167].

2.5.2.4 Software Packages

There are software packages available for measuring program comprehension; for example, Jadud's tool [71, 72, 73] and the ClockIt tool [52, 112]. These particular packages are represented as plug-ins for certain programming environments, in particular IDEs. However, there has been a case where related software was enabled in a command line environment [117]. These software packages can measure a student's program performance with criteria such as: date/time of compilation attempt, the number of compile/execution attempts, total number of lines of code, grade on assignment(s), total amount of time spent on the assignment etc. However, measuring the time spent on an assignment can also be applied to the efficiency of an environment.

2.5.3 Comprehension – Understanding Programming Procedures

A novice's awareness and understanding of programming procedures must also be considered when measuring program comprehension. These procedures consist of writing, compiling, linking, executing, and interpreting. As further discussed in Chapter 4, certain features within a programming environment may influence a novice's perception as well as

understanding of these procedures. It may be helpful for novices to manually perform each procedure while harmful for them to be exposed to “shortcuts” when conducting the same behavior.

Anecdotal evidence indicates that command line programming helps a programmer with understanding programming. One reason could be that command line environments use a lower level abstraction for programming than visual environments. Typically, visual environments, in particular IDEs, tend to consolidate programming procedures into one action that in many cases can be conducted with one button click.

2.5.4 Efficiency

Efficiency measures the ability for a programmer to write, compile, debug, and execute a program in a minimum amount of time. This measure determines if programming environments enable a programmer to perform a task in the minimum time frame necessary without losing the intended effect of program understanding. The following sections discuss two approaches: *Keystroke Level Model* and *Time on Task*.

2.5.4.1 Keystroke Level Model

Card, Moran, and Newell introduced the Keystroke Level Model to predict the potential amount of time for an expert user to execute a task [24]. Executing a task is composed of various physical motor operations. The total amount of time used to keystroke from a keyboard, T_K , equals the number of keystrokes, n_K , times the time per keystroke which is t_K ; $T_K = n_K * t_K$. When pointing to an object with a mouse, Fitt’s Law, $T_{pos} = K_0 + I_M \log_2(D/S + .5)$, can be applied. T_{pos} , represents the time to move the pointer to a target of size S which lies a distance D

away. K_0 represents the time for the hand to grasp the mouse and press the button to perform an intended action. T_{home} calculates the time for the user's hand to move from the keyboard to the mouse and back [24]. According to the Keystroke Level Model, it takes on average 1.10 seconds to point to an object with a mouse and another 0.20 seconds to click on that object [149]. *Note: The Keystroke Level Model and Fitt's Law only reflect the behavior of expert users.*

In order to accurately measure keystroking and mouse pointing, a user's mental preparation has to be measured. To further explore the importance of mental preparation, one must understand how the human mind functions. The Model Human Processor is an example tool for exploring the human mind [24]. According to this model, there are three main processors that define how the human mind behaves: *Perceptual Processor*, *Cognitive Processor*, and *Motor Processor*. The perceptual processor enables the human mind to detect the appearance of an object seen or heard and transmits it to the cognitive processor. The cognitive processor decides the appropriate response that the human should make based on the information retrieved and then passes the response to the motor processor. The motor processor is responsible for executing the appropriate human action given by the cognitive processor [24].

On average, it typically takes 1.35 seconds for a user to mentally prepare to do something [149]. Since programming typically involves a user interacting with a computer screen, it may take a little longer for the human mind to process this information. Nielsen states that users typically read 25% slower from a computer screen than from physical paper [111]. Overall, applying the Keystroke Level Model will require three actions for calculating the amount of time to perform a task using a programming environment; keystroking, positioning/pointing, and mental preparation: $T_{execute} = T_K + T_P + T_M$.

2.5.4.2 Time on Task

Time on task is a common approach for measuring efficiency. It can measure how efficiently a programmer can write, compile, debug, and execute a program. DePasquale [41] used this approach by subtracting the time of a student's last compilation/execution attempt from their first. Software packages can be instrumented to measure time on task; for example Jadud [71, 72, 73] and ClockIt [52, 112].

2.5.5 Usability

The usability of an environment helps determine how well a user can perform a certain task successfully, efficiently, and effectively. Sharp et.al defines usability as a way to ensure that interactive environments are *easy to learn, effective to use, and enjoyable from the user's perspective* [149]. Usability lays the foundation for whether a software tool is compatible for users based on certain criteria. Seffah et.al provides a table that lists different criteria for measuring usability [145]. In this table, Constantine and Lockwood, Shneiderman, Nielsen, Preece, and Shackel provide their respective criteria for measuring a tool's usability. The criteria are closely related to each other in the form of learnability, efficiency, reliability, memorability, and subjective satisfaction [23, 145]. Shneiderman represents these criteria as: time to completion, error rate by the user, time to learn, retention over time, and subjective satisfaction [149, 150].

Some of these criteria can be measured through other measures for environment evaluation. For example, efficiency is measured by time to completion while engagement is measured by subjective satisfaction. The remaining criteria can be considered attributes of usability. Time to learn measures how long it takes a user to learn how to use an environment.

Error rate indicates the number of errors made by students when using their respective environments. Retention over time conveys the easiness for the user to retain the essentials for using an environment.

2.6 Summary

This chapter discussed programming at early stages. The literature review and research background detailed issues and concerns regarding novices and their experience with programming. Appropriate paradigms, languages, and environments were the focal areas for discussion. In addition, appropriate measures for evaluating programming environments were discussed. Later chapters discuss studies involving environment evaluations while applying these measures. The environments involved in these studies have varying levels of feature assistance. The next chapter (Chapter 3) provides a detailed discussion about the variation of assistive features within programming environments.

3. FEATURE SET VARIATION

The structural make-up of programming environments varies. This variation is influenced by the different feature sets within programming environments. Section 3.1 discusses the variation of assistive features in programming environments and their potential effect on novice programming. Section 3.2 explores how certain styles of coding can influence a novice's understanding of programming. Section 3.3 discusses the familiarity of features and their potential of influencing a novice's perception of programming environments along with general procedures for programming.

3.1 A Continuum of Feature Sets

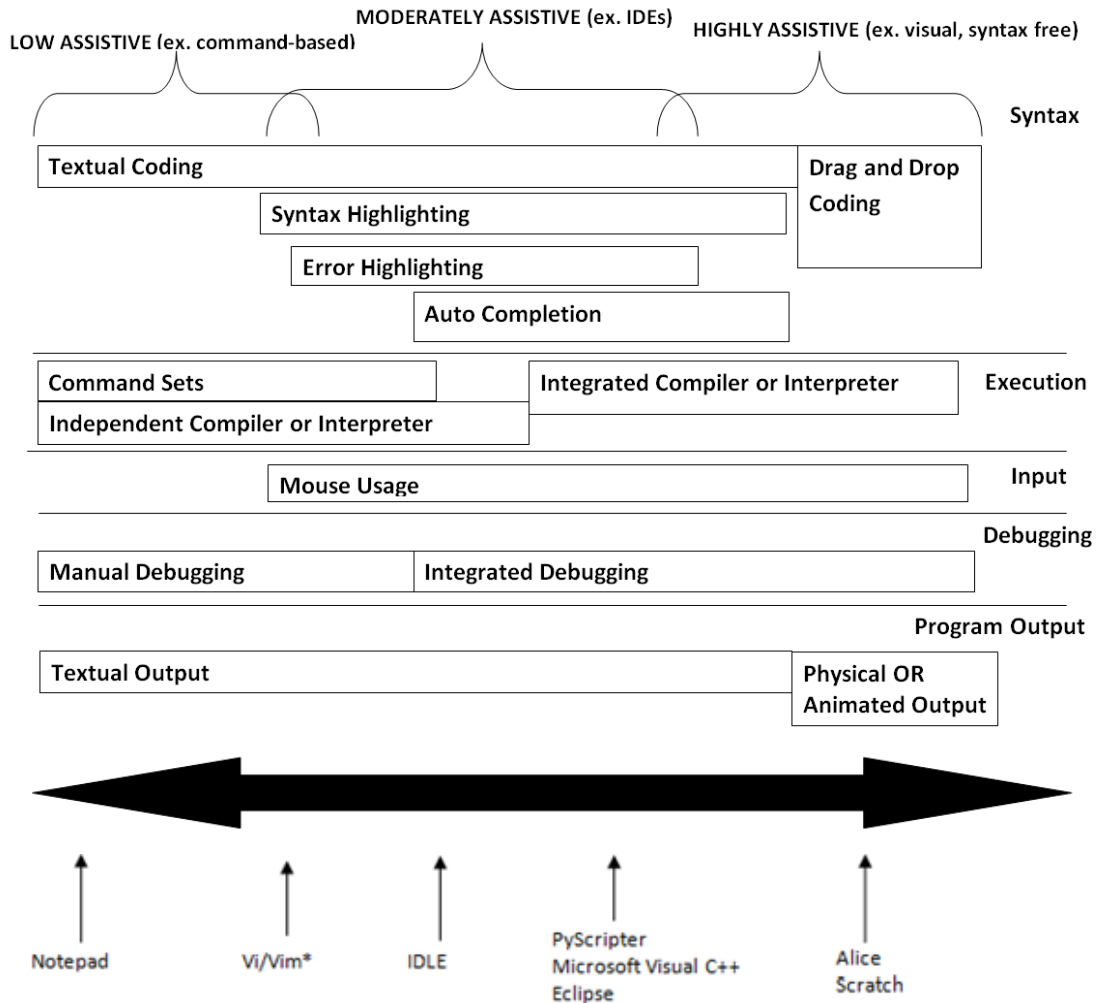


Figure 1: Programming Environments: Feature Sets [42]

**Feature set can readily be altered*

Figure 1 illustrates a continuum of basic feature sets that can be seen in programming environments [42]. Feature sets enable programming environments to provide *low assistance*,

moderate assistance, or *high assistance* to a programmer. Detail descriptions about each category are discussed in the following sections. Specific programming environments are aligned below the continuum based on their default feature sets. There are cases where individual features can be enabled or disabled within environments (*notice the asterisk*). This can alter an environment's behavior, which can also cause an environment to shift either left or right on the continuum.

3.1.1 Low Assistive Environments

Environments that are in this category typically possess basic essential features for programming [42]. Some of these environments may only provide the user with an editing window and a window for compilation/execution or interpretation. These environments typically allow the user to perform textual coding, command usage, and manual debugging. Users depend on some independent compiler or interpreter to run a written program. The feedback of a program is usually textual. Example environments that provide low assistance are plain text editors and text editors with very limited features. Example features that may be seen in low assistive environments are syntax highlighting and mouse usage. The VI/VIM editor, for example, provides the option of enabling syntax highlighting and mouse usage for programming [42].

Because of limited feature assistance, low assistive environments can provide the user with more flexibility for programming. Due to this flexibility, low assistive environments can be used to teach programming at any stage of a CS curriculum. In contrast, these limitations may also impose a higher learning curve for the environment itself, especially for novice programmers [42].

3.1.2 Moderately Assistive Environments

Environments that make up this category can provide a larger quantity of assistive features for programming [42]. Some of these features consist of syntax highlighting, error highlighting, auto completion, mouse usage, integrated compilation/execution (or interpretation), and integrated debugging. Usually, these environments can also provide textual feedback. There are some full-featured environments that possess similar traits seen in low assistive environments. These traits include: command sets, independent window for compiling/executing (or interpreting), and manual debugging. Example environments that fall into this category are rich-featured editors, intermediate and advanced/commercial IDEs [42].

In many cases, moderately assistive environments are able to provide users the flexibility for programming. Similar to low assistive environments, moderately assistive environments can be used to teach programming at any stage of a CS curriculum. In addition, their assistive features may help reduce programming tasks. However, there are environments in this category that could impose a high learning curve for novices because of their respective features [42].

3.1.3 Highly Assistive Environments

Similar to moderately assistive environments, highly assistive environments can possess a larger quantity of helpful features for programming [42]. Usually, these environments are built specifically to teach novices how to program. In many cases, the programmer is restricted to foundational programming concepts. Some highly assistive environments also require the user to perform drag and drop programming rather than syntax programming. In addition, physical or animated output can be used as an alternative to textual output. Example environments that

represent this category are graphical environments like Alice and Scratch and pedagogical IDEs [42].

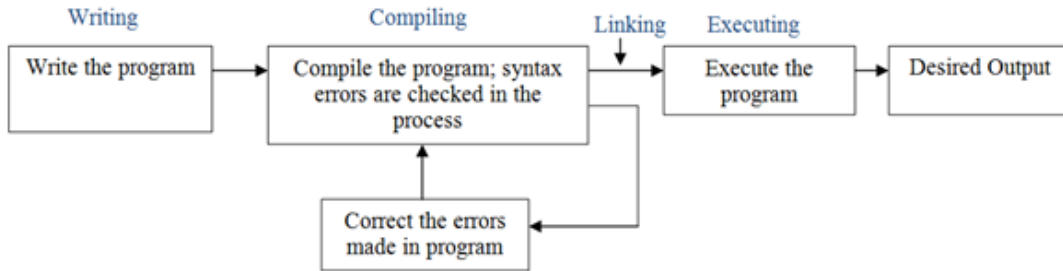
Highly assistive environments can withhold the flexibility of programming at any level. These environments may however impose a lower learning curve than low and moderately assistive environments. They are usually intended for early stages of programming. Therefore, novices would eventually have to transition from such environments to less assistive ones to round out their skill set [42].

3.2 Syntax vs. Drag and Drop Programming

Coding paradigms can provide an influential effect on how novices experience and understand programming. The difference between syntax and drag and drop programming is one factor. As seen in Figure 1 (continuum – last section), low and moderately assistive environments primarily permit textual coding or syntax programming. Syntax programming enables a novice to write, compile/interpret, and execute code. This process exposes novices to syntax errors and requires them to make the necessary corrections. At the same time, novices are placed in a position to develop an understanding of these underlying factors for programming. These factors can be seen in Figure 2, which provides the typical procedure for syntax programming.

Drag and drop programming, on the other hand, places more inferences on programming logic. In many cases, novices are not exposed programming syntax and errors. Instead they are dragging variables, functions, and methods, from their respective windows into the main body of an environment. Another constraint is that novices are prohibited from learning the procedures for compiling, linking, and executing a program. Rather, the novice removes and discards code while using mouse clicks to execute a program. Figure 3 provides a typical procedure for drag and drop programming.

Syntax Compiling Environments



Syntax Interpreting Environments

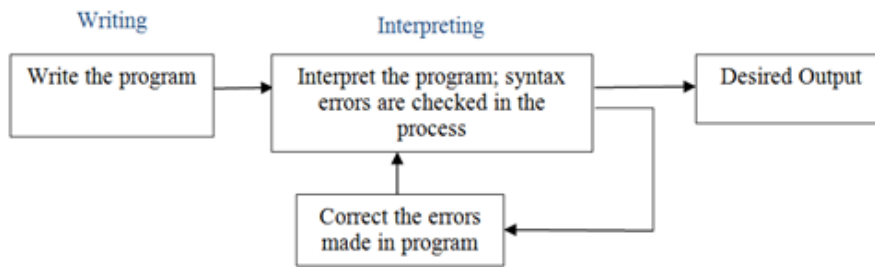


Figure 2: Outline of Syntax Programming

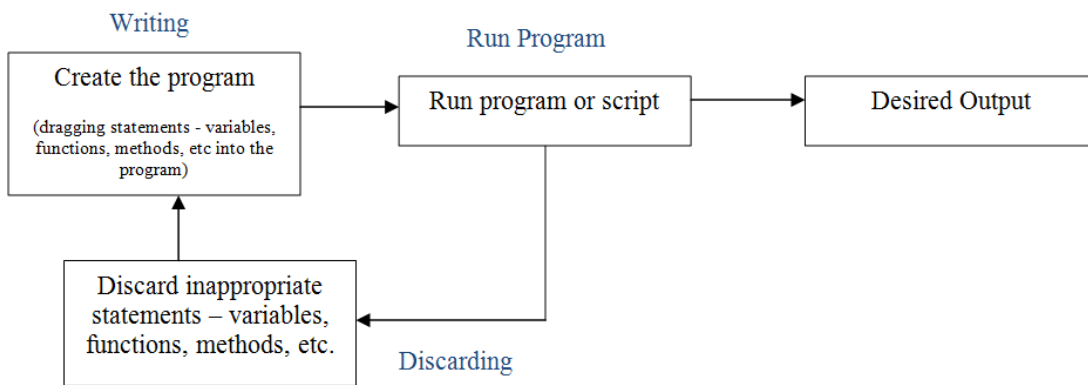


Figure 3: Outline of Drag and Drop Programming

Another factor that could influence how novices understand programming is the contrasting procedures between visual and command line tools. Chen and Marx [26] believed that a command line tool possibly provides students with a better understanding of programming. They also note that wizards (or features) for certain IDEs may prevent students from the same level of understanding.

Figures 4 and 5 illustrate the difference in the typical procedures for command line programming and IDE programming. When conducting command line programming, typically the user is restricted to command sets for writing, compiling/linking (or interpreting), and executing a program. During this process, the user has to manually perform each procedure using commands to obtain the output of a program (Figure 4). In contrast, IDEs can provide the user with a “shortcut” for performing the same procedures (Figure 5). In many cases, the user can complete this task with one button click, which triggers a compilation, linkage, and execution automatically. However, this shortcut may rob the user of fully understanding programming procedures, which is also important for a novice to understand. This difference is further discussed in Section 3.3.

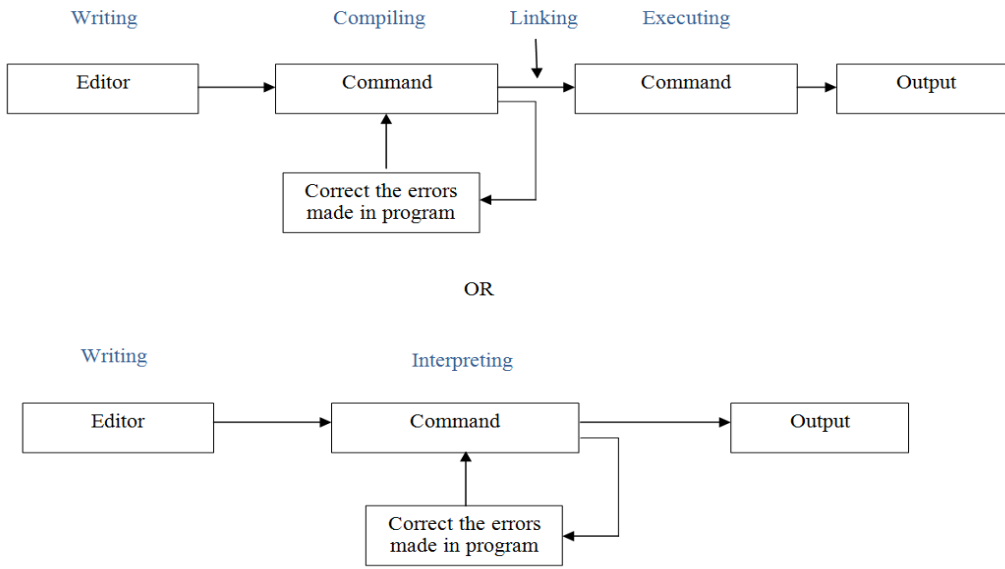


Figure 4: Outline of Command Line Programming

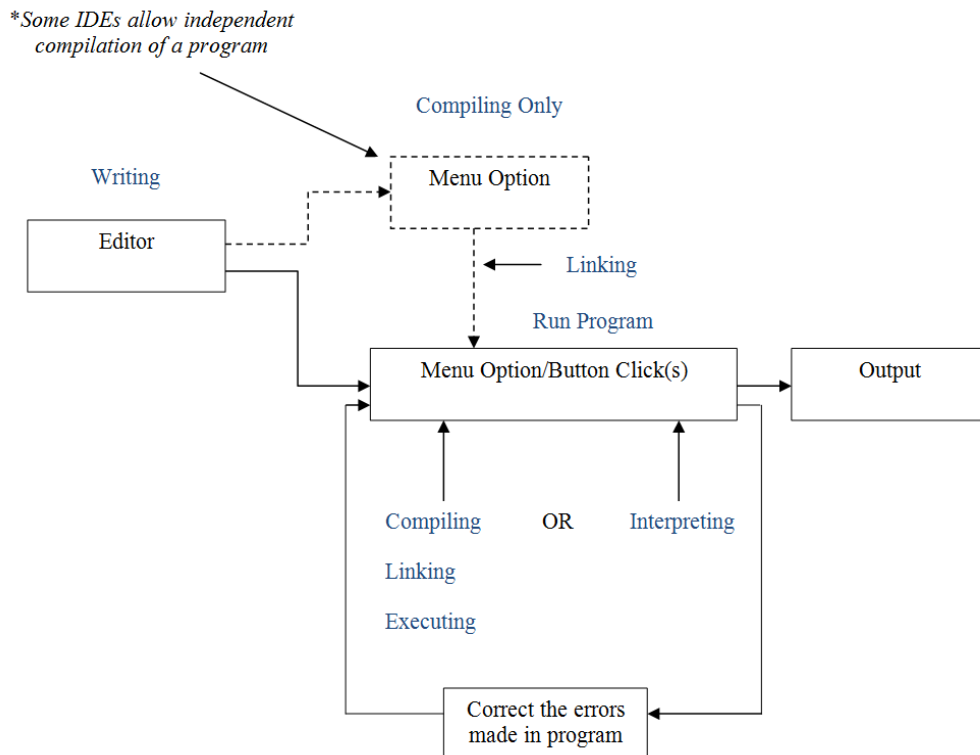


Figure 5: Outline of IDE Programming

3.3 Familiarity – Consistency and Affordance of Feature Sets

Familiarity with certain features could affect a novice’s perception of programming as well as the programming environment. WIMP (window, icon, menu, and pointing device) software tools like Microsoft Office, Internet Explorer, Safari, and iTunes provide service to end-users with different levels of computational experience. These tools also possess relative similarities to each other.

An appropriate term for this is *consistency*. According to Sharp et al., consistency is “designing interfaces to have similar operations and use similar elements for achieving similar tasks” [149]. Feature consistency can be seen in many visual environments, in particular IDEs. Figures 6 and 7 compare the consistency of menu features between various visual environments (both moderately and highly assistive respectively).

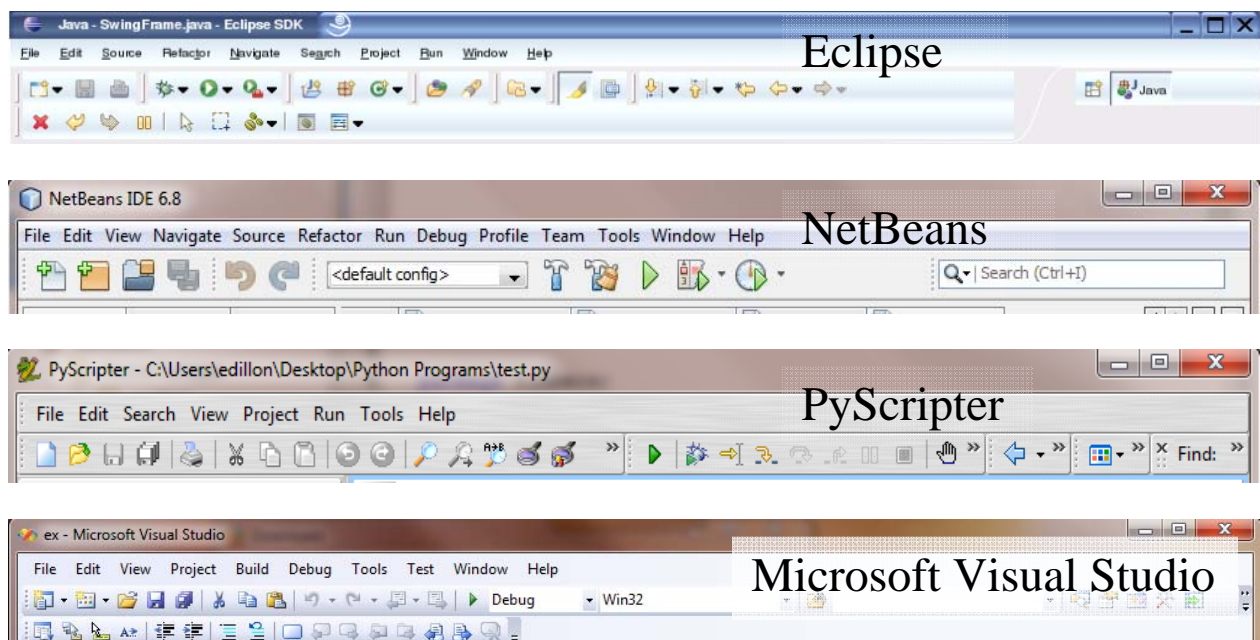


Figure 6: Advanced IDEs: Moderately Assistive [43]

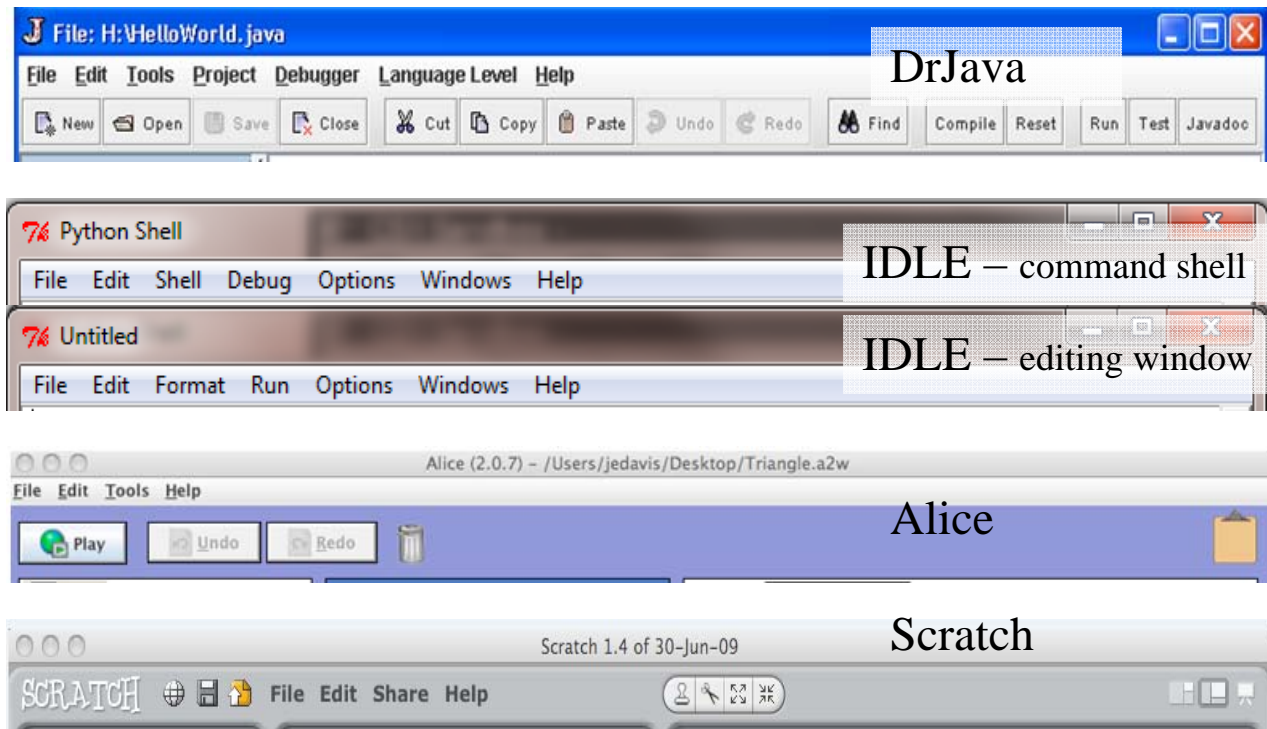


Figure 7: Pedagogical IDEs/ Graphical Environments: Highly Assistive [43]

Visual environments are typically built using WIMP features [43]. Each environment has a window interface that possesses different icons and menus for programming. The mouse is also used for navigating through these environments. As seen in Figures 6 and 7, each environment possesses the common attributes of a typical WIMP interface.

Companies who build different software tools tend to use consistent features for all of their software. For example, Microsoft is known for the creating Visual Studio and Word. The features for both tools are closely related. Figures 8 and 9 provide an example of consistency between Microsoft Visual Studio 2008 and Microsoft Word 2003 due to their similar attributes and structural make-up.

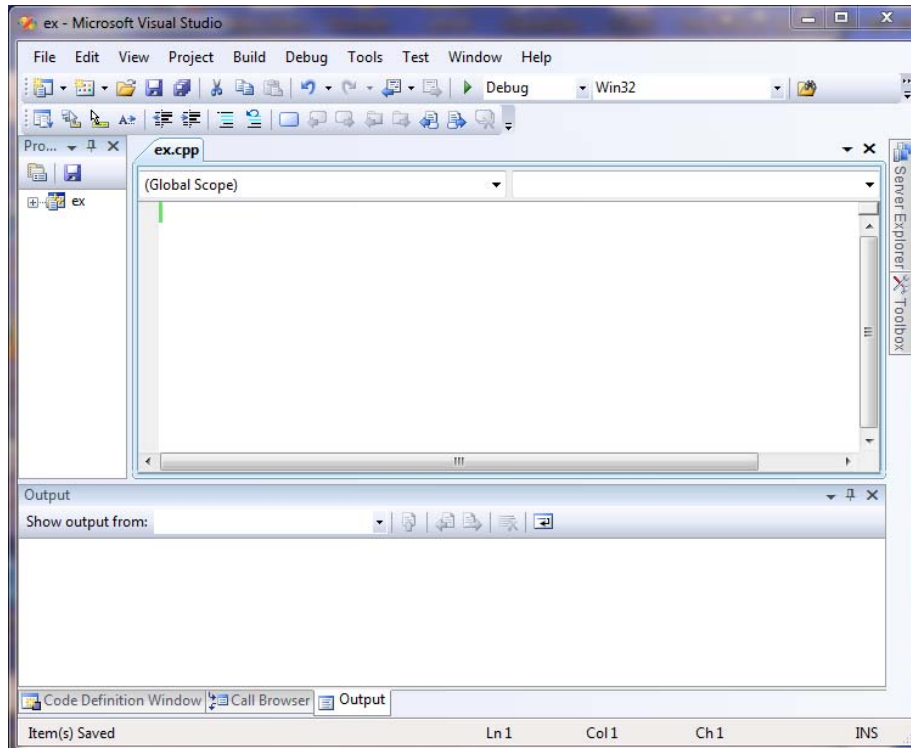


Figure 8: Microsoft Visual Studio 2008

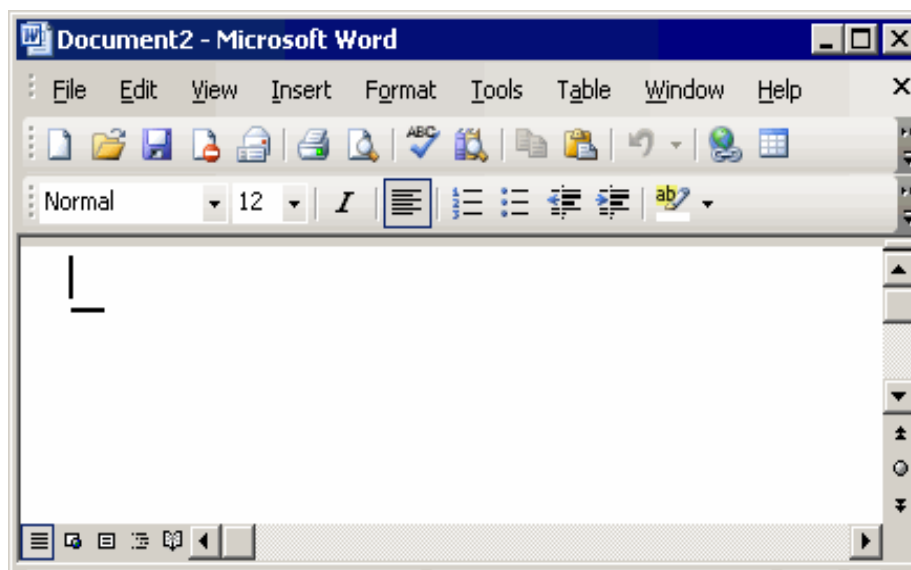


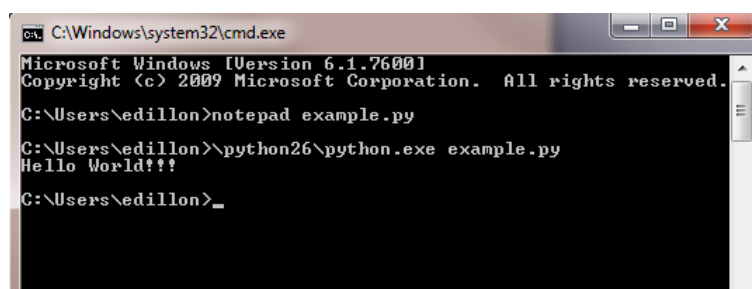
Figure 9: Microsoft Word 2003

The features within visual environments also provide the user with *affordance*. Affordance refers to an attribute or feature that gives the user “a clue” about how to use a particular object [149]. Today’s modern software tools possess attributes in forms of button

widgets, menu bars, icons, etc., that enable the end-user to understand a particular functionality. In Figures 6 and 7, each visual environment provides words, symbols, and icons to assist the programmer with understanding a specific function.

Prior to being exposed to programming, it is likely that novices have encountered modern WIMP software tools to complete some task. Examples tasks include surfing the web, online chatting, writing a term paper, or listening to music. Programming environments with similar consistency and affordance could be seen as familiar for novices who are learning to program. Such familiarity could influence an increase in a novice's comfort level with programming.

Another concern is whether environments with less consistency and affordance such as command terminals (Figures 10 & 11) could impose the same effect. With the exception of rich-featured text editors (Figure 12) and editing windows (Figure 13), novices are restricted from most menu options, button widgets, mouse usage, and the affordance of icons when introduced to command line programming. Instead, they are required to use command sets for navigation. By being exposed to less familiar features, it is possible that novices are forced out of their norm for understanding computer software. Rather, they have to adapt to a new norm for computing. This adaptation could also become more challenging if novices are required to move to a less familiar platform [43].



```
C:\Windows\system32\cmd.exe
Microsoft Windows [Version 6.1.7600]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Users\edillon>notepad example.py
C:\Users\edillon>python26\python.exe example.py
Hello World!!!

C:\Users\edillon>_
```

Figure 10: Command Terminal for Windows7 [43]

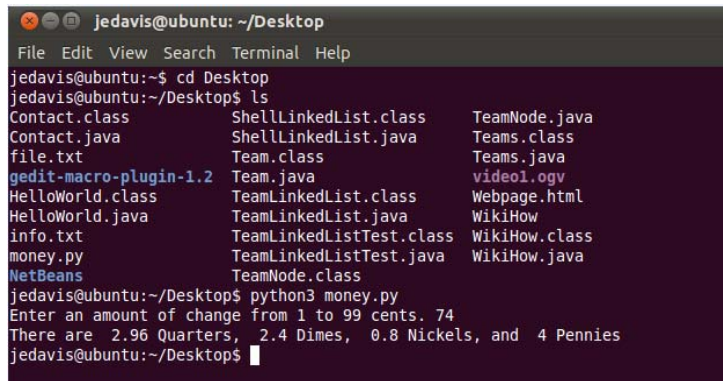


Figure 11: Linux Command Terminal - Ubuntu 10.10 [43]



Figure 12: JEdit Text Editor [43]



Figure 13: Notepad (editing window) [43]

On the other hand, moving novices into unfamiliar territory for their first programming experience could be beneficial. It is possible that their perception of software and computers in general are enhanced. Their perception of programming could have the same effect. It is believed that command line environments help novices obtain a better understanding of programming [26]. When novices are taught programming via a command line, they are using a lower level abstraction for navigation. Most of this navigation is manual. Command line environments require a user to manually compile/execute, interpret, or debug a written program. Typically, users cannot bypass one procedure without completing the other (see Figure 4). For many visual

environments, this is not the case. Typically, programmers can conduct the same behavior with one or more button clicks (see Figure 5).

If novices are exposed to a familiar norm when learning to program, it is possible they could develop a false perception of programming. For example, a novice who is introduced to programming using a visual environment with familiar features may not see programming as distinctly different from using a word editor. Typically, visual environments expose users to a higher level of abstraction for navigation. The same can be said about novices who are taught programming via most visual environments. This high level exposure could make novices susceptible to false impressions rather than obtaining a true understanding for programming. For example, a novice could get the impression that the “Run” button magically makes their program work. This single widget could deprive them of understanding compilation/execution (or interpretation) [43].

3.4 Summary

This chapter discussed the variation of feature sets within programming environments. Feature consistency and affordance can be helpful as well as harmful for novice programmers. In many visual environments, these features give the novice a sense of familiarity for using certain environments. In contrast, these features could weaken a novice's mental model for understanding basic programming procedures. On the other hand, environments with less familiar features may help a novice better understand the underlying concepts of programming. As mentioned in Chapter 2, there is only anecdotal evidence to support either case. Chapters 5 and 6 detail respective studies that potentially provide answers to these suppositions. Chapter 4 introduces the first environment evaluation study in relation to this research.

4. ALICEVILLE OUTREACH

This chapter provides an example study for applying the measures discussed in Chapter 2 to evaluate programming environments and their effect on novices. This study was conducted at Aliceville High School in Pickens County, AL during the Spring 2011 semester. This study was part of an outreach for exposing underrepresented minorities to the field of Computer Science.

Aliceville High School is predominantly African American. All the participants in this outreach were African American with the exception of one who was Native American. According to the *Journal of Blacks in Higher Education*, roughly 5% of computer scientists are African American. At the graduate level, the representation of advanced degrees awarded to African Americans in this field is even less [81]. The objective was to expose these students to the ideal of pursuing computer science as career choice.

The participants were exposed to robotic programming using PREOP (Providing Robotic Experiences through Object-based Programming). PREOP is a software package developed at the University of Alabama. This environment consists of the Alice environment (Figure 14a) and an IRobot Create (Figure 14b). Alice provides a virtual image of the IRobot Create for running simulations. It also provides a real mode interaction that allows physical interaction with the robot via a Bluetooth device and a BlueCove library.

The outreach was conducted one day per week and lasted for roughly five weeks. A different CS1 concept was discussed each week. These concepts include decomposition, sequencing, decisions, boolean expressions, and variable usage. These students worked in groups of 3-4 every week.

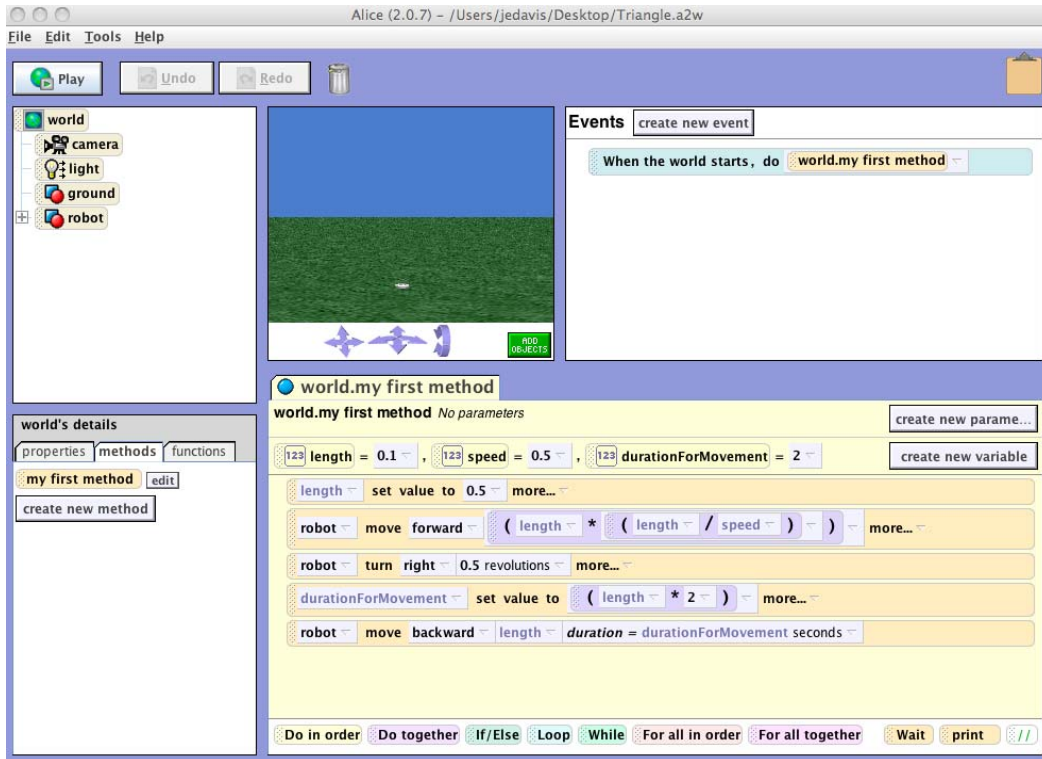


Figure 14a: Alice version 2.0.7



Figure 14b: I-Robot Create

During the final week, the students received a survey about their experience with PREOP. Questions involving ease of use (*usability*), self-efficacy (*engagement*), and Pennington's model

(*comprehension*) were asked in this survey. Efficiency was not measured since there was no other environment involved during this outreach. The next sections detail the results from this study.

4.1 Demographics

There were twenty African American students and one Native American student who participated in this study. The number of male and female participants were 9 (43%) and 12 (57%) respectively (Figure 15). The participants were all juniors (Figure 16). Every student indicated not to have any programming experience prior to this outreach.

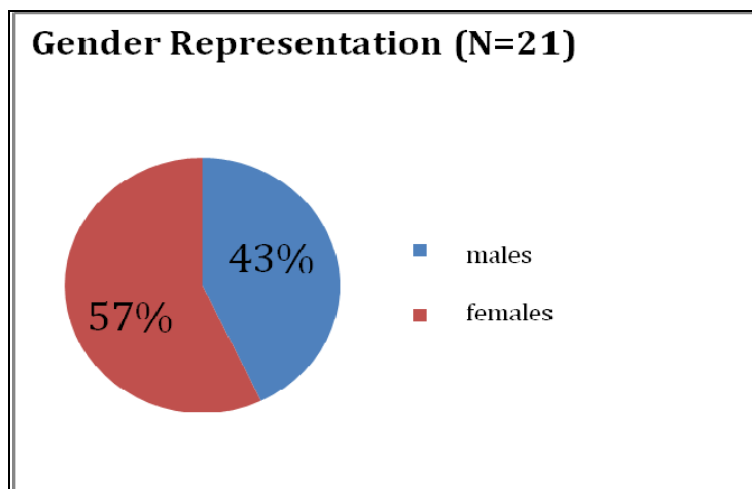


Figure 15: Gender Representation

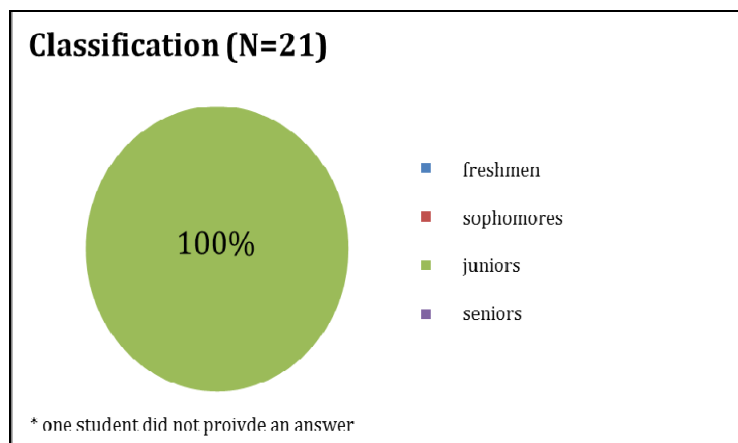


Figure 16: Student Classification

4.2 Usability

This section discusses the results of measuring PREOP's usability. Measures consist of ease of use, reliability, frustration, and future usage. Additional questions were asked to measure comfort for using robots and general interest in computer science in order to control for any confounding factors. The following subsections details each measure.

4.2.1 Ease of Use

This question was based on a 5-point Likert scale, ranging from *not easy at all* to *absolutely easy*. Majority of the students indicated *mostly easy* (43%) or *50/50* (43%). The percentage for *absolutely easy* was 4% while *slightly easy* had a 10% response. There were no responses of PREOP being *not easy at all*. Therefore, many of the students felt that PREOP was either somewhat easy or mostly easy to use (Figure 17).

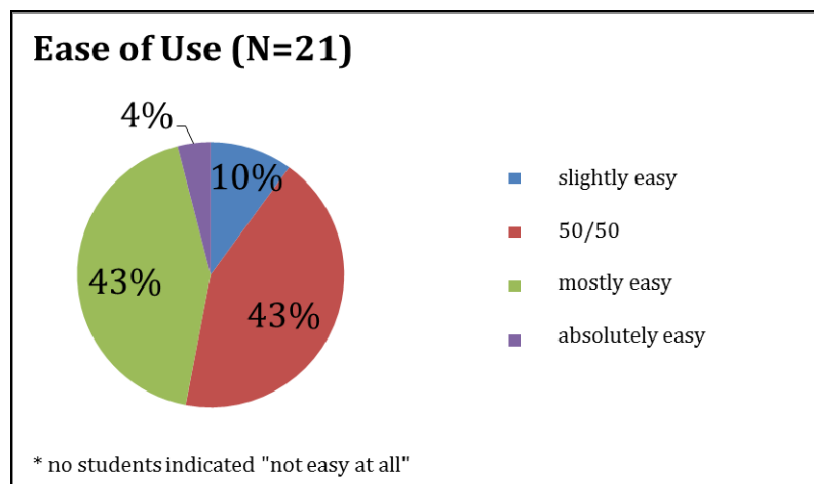


Figure 17: PREOP - Ease of Use

4.2.2 Reliability

This question was asked in conjunction with the reliability of robots. There were four possible responses for this question *no problems*, *a few problems*, *many problems*, or *did not work well*. Majority of the students indicated *a few problems* (57%). The percentage of responses for *no problems* was 33% while *many problems* had a 5% response. There were no responses for *did not work well*. Therefore, many of these students experience a few problems with PREOP while programming (Figure 18).

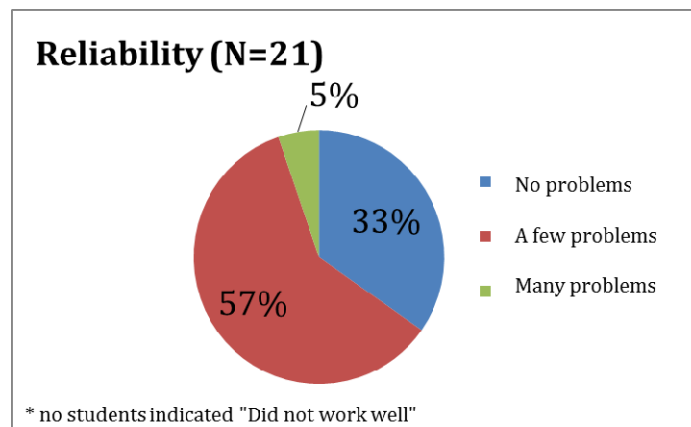


Figure 18: PREOP - Reliability

4.2.3 Frustration

This question required a yes/no response. Majority of the students indicated a response of *no* (75%) for being frustrated with the process of programming in PREOP. The percentage of *yes* responses was 25%. Therefore, many of these students were not frustrated with using PREOP to program (Figure 19).

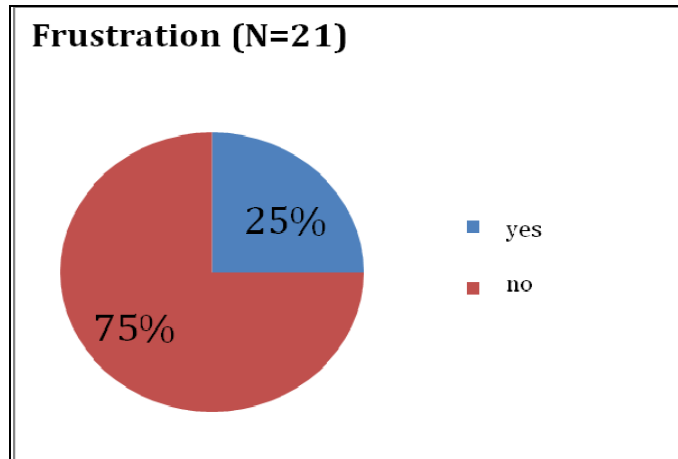


Figure 19: PREOP - Frustration

4.2.4 Future Usage

This question was asked in two parts: *Would you use PREOP in the future?* and *Would you use PREOP as a hobby or extracurricular activity?* For the former question, majority of the students indicated a response of *50/50* (45%). A response of *not at all* was 20%, *absolutely yes* was 15%, and *slightly yes* and *mostly yes* was 10% respectively. One student did not provide an answer for this question. For the latter question, majority of the students indicated a response of *not at all* (32%). A response for *slightly yes* was 26%, *50/50* was 21%, *mostly yes* was 16%, and *absolutely yes* had a 5% response. Therefore, many of these students felt neutral about using PREOP for future programming. However, a slight majority indicated no interest for using PREOP during leisure time (Figure 20).

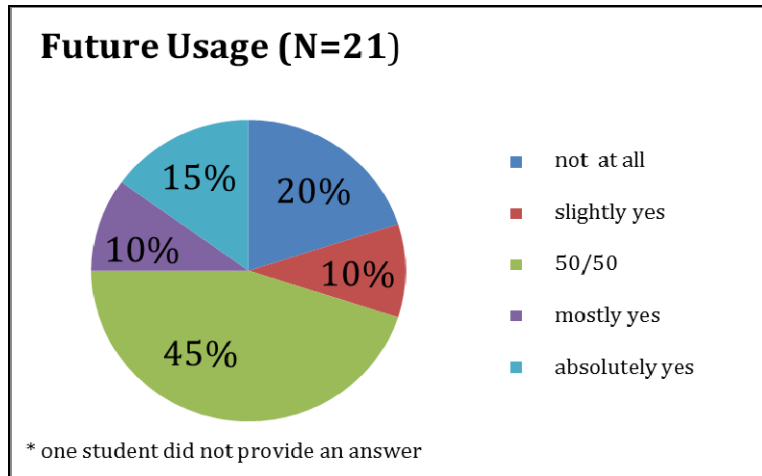


Figure 20: PREOP - Future Usage

4.2.5 Comfort with Programming Robots

This question was based on a 5-point Likert scale, ranging from *not comfortable at all* to *absolutely comfortable*. Majority of the students indicated to be *absolutely comfortable* (33%) or *50/50* (33%) with programming robots. The percentage of responses for being *mostly comfortable* was 24% while 10% indicated to be *slightly comfortable*. There were no responses for being *not comfortable at all*. Therefore, many of these students felt either somewhat or absolutely comfortable with programming robots (Figure 21a).

In addition, an open-ended question was asked to gather more information about their comfort with programming robots. The students' responses were categorized into three categories *positive*, *non-positive*, or *no response*. Majority (52%) gave positive reasons for their level of comfort, 33% had a non-positive reason, and 14% gave no response for this question. Many of these students provided positive feedback about using robots. Some of their responses pertained to simplicity of usage, comfort and amusement, and excitement for learning something new. For the students who gave non-positive feedback, they responded as being uncomfortable, confused, or simply not interested in computers and robots (Figure 21b).

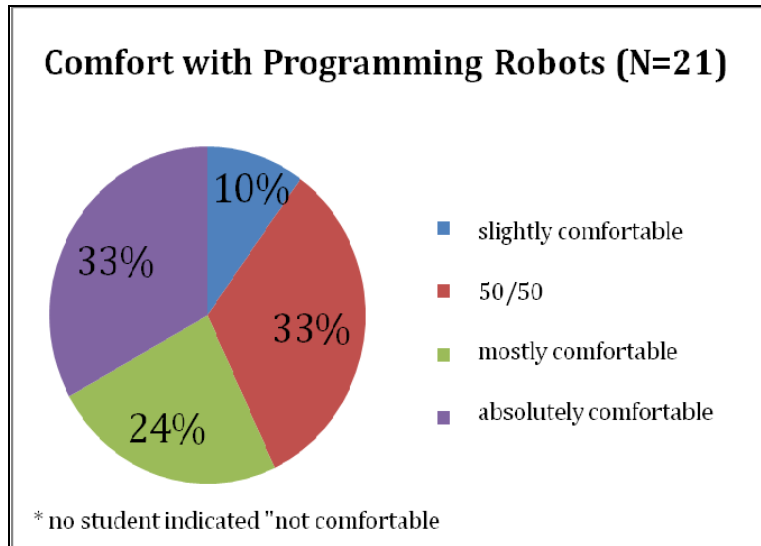


Figure 21a: PREOP – Comfort with Programming Robots

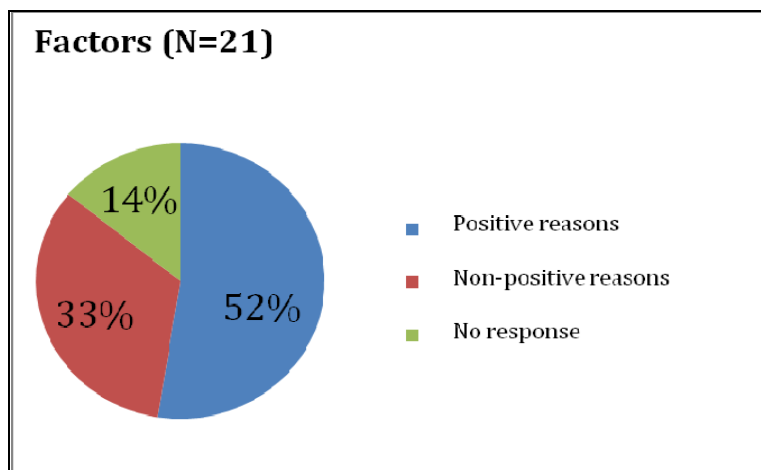


Figure 21b: PREOP – Comfort with Programming Robots (Factors)

4.2.6 Interest in Computer Science

This question was based on a 5-point Likert scale, ranging from *not at all* to *absolutely yes*. Majority of the students indicated *mostly yes* (29%) and *slightly yes* (29%). The percentage of responses for *not at all* was 19%, *absolutely yes* was at 14%, and 10% indicated *50/50*. Therefore, many of these students were either mostly or slightly interested in pursuing a career in computer science.

An open-ended question was asked to gather more information about their interest (or no interest) in computer science. The students' responses were categorized into three categories *interested*, *not interested*, or *no response*. The results showed an equal percentage (43%) for students being interested as well as not interested in computer science as a career. The response percentage for *no response* was 14%. Overall, there was a mixed response about pursuing a career in computer science. Students that were interested, indicated to some affect that they enjoyed working with computers. Responses for those who were not interested consisted of boredom, dislike towards computers and technology, or interest in other fields (Figure 22).

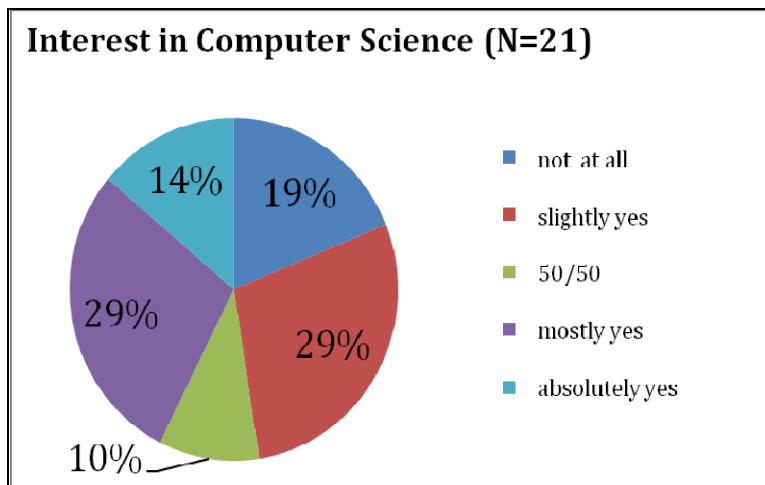


Figure 22: PREOP – Interest in Computer Science

4.3 Self-Efficacy

The students' self-efficacy was measured by 21 questions. Each question was based on a 5-point Likert scale, ranging from *not at all* to *absolutely yes*. The mean score for all of the students was 63.81 out of a maximum possible score of 105 (Table 3). The average response for each question was a 3.32, which indicates that as a whole the students' self-efficacy for programming in PREOP was slightly above neutral (or 50/50). Additional measures were conducted to compare gender differences by using T-Tests. Four of the 21 questions indicated a significant difference:

- *the female participants (as a whole) were less comfortable with programming robots than their male counterparts ($p < 0.05$);*
- *the female participants (as a whole) were less confident about completing a program once someone else helped them get started than their male counterparts ($p < 0.05$);*
- *the female participants (as a whole) were less confident about completing a programming project if they got stuck than their male counterparts ($p = 0.05$);*
and
- *the female participants (as a whole) were less confident about the possibility of solving a problem in two different ways and getting two different results than their male counterparts ($p < 0.05$).*

Table 3: Self-Efficacy Descriptive Data

Participants	N	Mean	StDev	Min Score	Max Score	Possible Score	Average Response for each question
Together	21	63.81	14.83	40	90	105	3.32
Females	12	66.50	15.89	48	90	105	3.33
Males	9	60.22	13.34	40	87	105	3.31

4.4 Pennington's Model

Students were required to look at a snippet of code from PREOP and answer five questions that measured their understanding of elementary operations, control flow, data flow, program function, and program state for programming. Some students managed to provide five answers while others did not answer every question. Table 4 shows the actual number of students who answered each question. Table 5 provides percentages of the correct responses for each question. Results showed that majority of the students gave correct responses to each question except for the program's function; only 26% gave the correct answer.

Table 4: Number of Answers for each question

N = 21	Elementary Operations	Control Flow	Data Flow	Program State	Program Function
Number of responses for each question	18	17	16	17	19

Table 5: Correct/Incorrect Percentages

Elementary Operations	Control Flow	Data Flow	Program State	*Program Function
Correct: 100%	Correct: 65%	Correct: 69%	Correct: 76%	Correct: 26%
Incorrect: 0%	Incorrect: 35%	Incorrect: 31%	Incorrect: 24%	Incorrect: 74%

An abstract or functional mental model is required to understand a program's functionality. According to Adelson, typically expert programmers use functional information to

think about programming, while novices use concrete information [1]. The students only had five days of programming during this outreach. More than likely, they were thinking concretely when answering this question. It is also possible that those who provided the appropriate answer may have guessed correctly.

Similar actions could have occurred with the other questions. There were students who did not complete Pennington's model in its entirety. Due to their lack of programming exposure, it may have been challenging for them to understand the snippet of code. Another possibility could be that these particular students were confused by Pennington's model.

4.5 Discussion and Summary

The 21 participants had no prior programming experience. However, many of them believed that PREOP was fairly easy to use. This supports why many of them were not frustrated with programming. They also showed to be fairly comfortable with programming robots. Their self-efficacy for programming was slightly above neutral. Pennington's model showed that majority of these students understood the programming concepts. The only exception was their understanding of a program's overall functionality. It is possible more programming experience was required in order to understand this concept. Overall, this outreach exposed underrepresented minorities to the possibility of a career in computer science. However, majority of these students showed a mixed interest about pursuing this field for different reasons.

This study evaluated a highly assistive visual environment and its effect on novice programming at the high school level. This study required adjustments for the measures. One adjustment was the exclusion of an efficiency measurement since only one environment was involved in this study. The usability and self-efficacy questionnaires were also modified to cater to a five-day outreach. The next chapter provides a detailed study that compares three programming environments (one low assistive, two moderately assistive) and their effect on novice programmers at the college level. All four measures were incorporated.

5. CS1 LABORATORY STUDY – PYTHON PROGRAMMING

Pears et.al concluded from their survey that new significant empirical results should be the focus when doing research on how to teach programming [119]. This chapter details a study conducted on students who were currently taking a CS1 Laboratory course during the spring 2011 semester [42]. The objective of this study was to measure the effect of programming environments with different levels of feature assistance on students in this course. CS160 is the CS1 lab course taught in conjunction with the first programming course (CS150). In CS150, students are taught Python using the VIM command line environment on the Linux platform. In CS160, students are introduced to robotic programming through PREOP that allows them to program real robots using syntax free, drag-and-drop procedures in Alice (*refer back to Chapter 4 for further details about PREOP*). CS160 has no prerequisites and two or three sections are usually offered per semester. Three sections were offered during this semester. For this study, each section received its own programming environment: Section 1 – IDLE, Section 2 - PyScripter, and Section 3 – Notepad. Each section is classified by their respective environment for the remainder of this section (*the IDLE group, the PyScripter group, and the Notepad group*).

5.1 Environments/Experiment Conditions

IDLE (Figure 23) was developed by Guido van Rossum in 1999 [141]. Since its initial development, this environment has evolved. IDLE is built using the Python language and the Tkinter GUI toolkit. It can be used on both the Windows and Linux platforms. The structural makeup of IDLE consists of two windows: a shell (for code interpretation) and an editor (for writing code). The shell window can also be used to write snippets of code for interpretation. IDLE's features include: syntax highlighting, auto completion, and integrated debugging.

PyScripter (Figure 24), also known as *Portable Python*, is a full featured IDE. Currently, PyScripter is only available for the Windows platform. The structural makeup of PyScripter consists of one window that integrates the editor, interpreter, and debugger. Some of PyScripter's features include: syntax highlighting, code completion, call tips, and a variables window [126].

Notepad (Figure 25) is a text editor on the Windows platform. For programming, Notepad can be used to write and edit code. Unlike many IDEs and other text editors, Notepad does not enable feature assistance such as syntax highlighting, auto completion, or auto debugging. The command prompt terminal can be used to run the Python interpreter and display the results from a written program.

Four measurements were used in this study. The Computer Programming Self-Efficacy Scale measured the self-efficacy of the students in regards to programming. Pennington's Model measured the students' programming comprehension. The Pennington's model surveys were based on one of the procedural programs (Program E) used in Ramalingam and Wiedenbeck's

study [131]. Time on task was also measured in this study. An additional survey was given to measure the environments' usability.

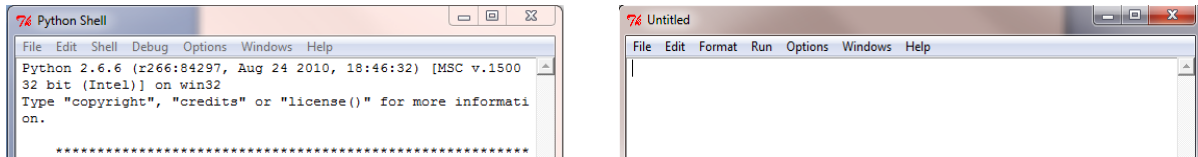


Figure 23: IDLE version 2.6.6

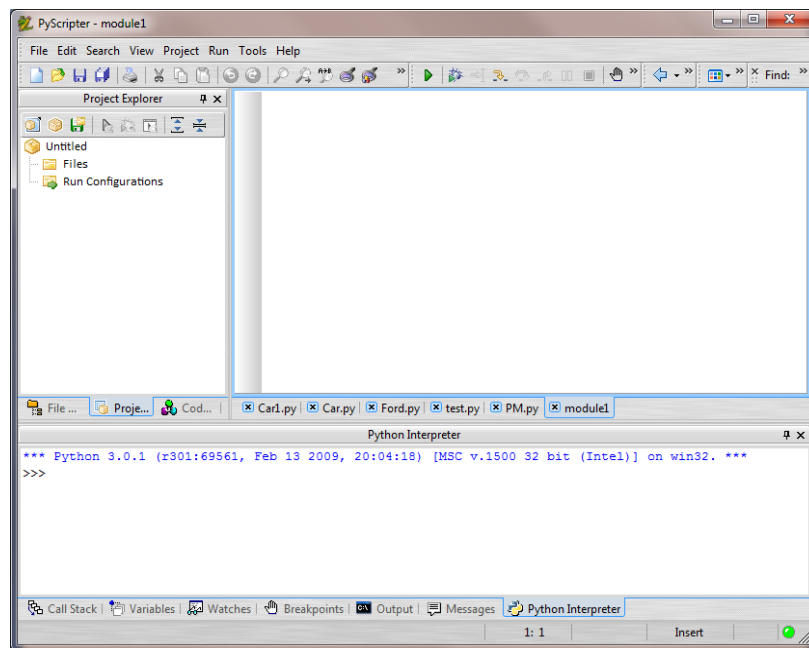


Figure 24: PyScripter version 1.9.9.6

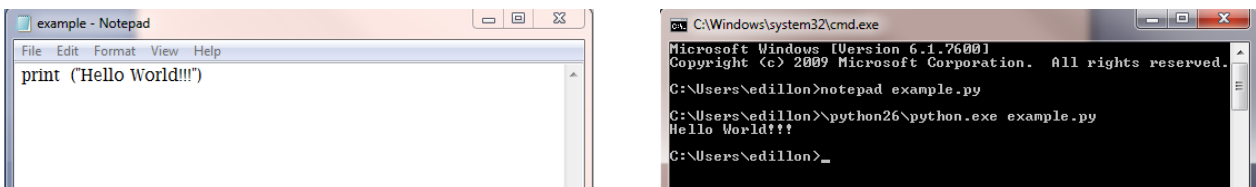


Figure 25: Notepad/Command Prompt (Windows7 version)

5.2 Demographics

The student representation for CS160 varied for each procedure. This occurred because students either arrived late to class or failed to follow the instructions closely during the study. The following demographics statistics represent a population of 94 students (Tables 6 - 7b). These statistics were calculated through one-way ANOVAs or T-tests.

The student representation consisted of different majors and classification levels. The PyScripter group in particular had more Electrical Engineering majors (42%) than Computer Science (24%). The PyScripter and Notepad groups had significantly more Juniors than the IDLE group ($p < 0.05$). Each group respectively had a significantly higher male enrollment than females ($p < 0.01$). However, the females were significantly more ($p < 0.01$) intimidated with programming than their male counterparts. The entire population was intimidated more by programming than computer science itself. This was true before taking CS160 ($p < 0.05$) as well as during the time of this study ($p < 0.05$). In regards to CS150, only 31% of the CS160 population were also taking or had already completed this course. However, the Notepad group (50%) had significantly more ($p < 0.01$) CS150 students than the IDLE group (13%); the PyScripter group had 31%. The IDLE group (40%) also had less prior programming experience than the PyScripter group (26%), which was also significantly less ($p < 0.05$) than the Notepad group (15%). When looking at current GPAs, the Notepad group had lower GPAs than both the IDLE ($p < 0.05$) and PyScripter ($p < 0.01$) groups respectively. For each section, the majority of the students were going to take another programming course during the following

semester: the IDLE group (80%), the PyScripter group (68%) and the Notepad group (62%). In addition, the majority of these students needed another programming course in order to graduate: the IDLE group (90%), the PyScripter group (82%), and the Notepad group (73%).

Table 6: CS 160 Demographics

Student Representation (N=94*)			
Major	Classification	GPA	Gender
Computer Science - 33% Electrical Engineering - 29% Computer Engineering - 15% MIS - 3% Math - 5% Other - 18%	Freshmen - 41% Sophomore - 32% Junior - 22% Senior - 3% Other - 1%	3.0 - 4.0: 59% 2.0 - 3.0: 36% 1.0 - 2.0: 4% First Semester (No GPA): 1%	Male - 74% Female - 26%
Programming Experience	Programming Intimidation (prior to CS160)	Programming Intimidation	Prior Experience with Other Environments (besides PREOP)
CS1 programming - 31% High School programming - 26% Another College Course - 18% No Experience - 26%	Yes - 44% No - 56%	Yes - 44% No - 56%	Yes - 49% No - 51%
Males Intimidated by Programming (N=70)	Females Intimidated by Programming (N=24)	Prior Experience with Visual and Command Line Environments (N=47)	Environment Mandatory for a course?
Yes - 33% No - 67%	Yes - 75% No - 25%	Visual - 43% Command Line - 57%	Yes - 35% No - 65%
Statistical Significance			
Programming Intimidation (Male vs. Female): T-tests showed a significant difference ($p < 0.01$).			
<i>*Number of responses before Time on Task was conducted.</i>			

Table 7a: Section-by-Section Demographics

Group	Major	Classification	GPA
IDLE (N=30*)	Computer Science - 37% Electrical Engineering - 27% Computer Engineering - 23% MIS - 7% Math - 7% Other - 7%	Freshmen - 57% Sophomore - 37% Junior - 7% Senior - 0% Other - 0%	3.0 - 4.0: 63% 2.0 - 3.0: 33% 1.0 - 2.0: 3% First Semester (No GPA): 0%
PyScripter (N=38*)	Computer Science - 24% Electrical Engineering - 42% Computer Engineering - 13% MIS - 3% Math - 3% Other - 18%	Freshmen - 32% Sophomore - 37% Junior - 39% Senior - 0% Other - 3%	3.0 - 4.0: 71% 2.0 - 3.0: 29% 1.0 - 2.0: 3% First Semester (No GPA): 0%
Notepad (N=26*)	Computer Science - 42% Electrical Engineering - 12% Computer Engineering - 8% MIS - 4% Math - 8% Other - 31%	Freshmen - 38% Sophomore - 19% Junior - 31% Senior - 12% Other - 0%	3.0 - 4.0: 35% 2.0 - 3.0: 50% 1.0 - 2.0: 12% First Semester (No GPA): 4%
Statistical Significance			
<p>Major: A one-way ANOVA was conducted ($p < 0.05$); A t-test showed a significant difference between PyScripter and Notepad groups ($p < 0.01$).</p> <p>Classification: A one-way ANOVA was conducted ($p < 0.05$); T-tests showed a significant difference between IDLE and PyScripter groups ($p < 0.01$) and IDLE and Notepad groups ($p < 0.05$) respectively.</p> <p>GPA: A one-way ANOVA was conducted ($p < 0.05$); T-tests showed a significant difference between IDLE and PyScripter groups ($p < 0.05$), IDLE and Notepad groups ($p < 0.05$), and PyScripter and Notepad groups ($p < 0.01$) respectively.</p>			
<i>*Number of responses before Time on Task was conducted.</i>			

Table 7b: Section-by-Section Demographics (CONT'D)

Group	Gender	Programming Experience	Programming Intimidation (prior to CS 160)	Programming Intimidation
IDLE (N=30*)	Male - 73% Female - 27%	CS1 programming - 17% High School programming - 17% Another College Course - 17% No Experience - 40%	Yes - 50% No - 50%	Yes - 50% No - 50%
PyScripter (N=38*)	Male - 74% Female - 26%	CS1 programming - 34% High School programming - 16% Another College Course - 24% No Experience - 26%	Yes - 45% No - 55%	Yes - 37% No - 63%
Notepad (N=26*)	Male - 77% Female - 23%	CS1 programming - 50% High School programming - 27% Another College Course - 8% No Experience - 15%	Yes - 35% No - 65%	Yes - 46% No - 54%

Statistical Significance

Gender (Male vs. Female): T-Test showed a significant difference in each group respectively ($p < 0.01$).

Programming Experience: T-Test showed a significant difference between IDLE and Notepad groups ($p < 0.01$).

Programming Intimidation (Male vs. Female): T-tests showed a significant difference for PyScripter ($p < 0.05$) and Notepad ($p < 0.05$) groups.

**Number of responses before Time on Task was conducted.*

5.3 Procedures

For all three sections of CS160, each study was conducted in the same order. To begin the study, each student received a self-efficacy survey. This survey consisted of 31 questions from the Computer Programming Self-Efficacy Scale. The responses were given on a 7-point Likert scale that ranged from *not confident at all* to *absolutely confident*.

After the survey, the students received an introductory lecture on the Python language. This lecture provided primary Python concepts that the students would need to complete the exercise. The objective was to expose the students to concepts of selection, information hiding, syntax, and semantics. The lecture began by introducing print statements and their functionality. The next topic was variable usage and assignment. At this point, the students were also introduced to the reserved keywords in Python. The following topic introduced mathematical operations. In particular, students learned the difference between division and modulus operations. The lecture concluded by showing students an example program using every topic. This program converted x number of minutes into h hours and m minutes remaining. The functionality of this program resembled the assignment that the students would be asked to write. After the lecture, the students received a demonstration on how to use their respective environment.

For their assignment, the students were required to write a small program that converted 700 days into y years, m months, and d days remaining. The students would write this program using their respective environment. During this process, the objective was to measure the students' time on task for writing the required program. For the IDLE group, a process

monitoring application was used to measure time on task. In order to access their time logs, the students first accessed the process monitoring application before using IDLE, and then remain logged onto their computers after completing the assignment. However, some students did not follow these directions closely which resulted in their respective time logs being lost. Therefore, the remaining two sections did not use the software. Instead these students were required to start at the same time and were required to raise their hands upon completing the assignment. The time on task for these sections was calculated by subtracting time of completion from the starting time. As part of this process, behavioral observations were conducted on the students. These observations were recorded on paper. Once a student hand was raised to indicate completion, their end time was also recorded on paper. After completing the exercise, the students were required to complete two final surveys: a Pennington's Model survey, and a usability survey. Two versions of Pennington's Model were issued to the students in order to prevent anyone from copying answers. The questions in Version 1 were identical to the questions from Ramalingam and Wiedenbeck's study [131]. Version 2 was a modified version of Version 1.

5.4 Results

5.4.1 Self-Efficacy

The mean self-efficacy score for the students was 114.85 out of a possible score of 217 (Table 8). The mean self-efficacy scores amongst the three sections (Table 9) were tested using a one-way ANOVA. The ANOVA showed a significance ($p < 0.01$). The ANOVA test was followed by T-tests to determine whether specific differences existed amongst the sections. The results from the T-test showed a significant difference between the IDLE and PyScripter groups ($p < 0.01$) as well as the IDLE and Notepad groups ($p < 0.01$) respectively. There was no significant difference between the PyScripter and Notepad groups. This indicated that students in the IDLE group were less confident in their programming abilities than their counterparts in the PyScripter and Notepad groups respectively.

Table 8: Self-Efficacy Descriptive Data for CS160 (N = 94)

Mean	StdDev	Min Score	Max Score
114.85	46.83	23	207

Table 9: Self-Efficacy Descriptive Data Amongst The Three Sections

Group	Mean	StdDev	Min Score	Max Score	N
IDLE	88.30	38.91	23	177	30
PyScripter	125.63	49.57	34	207	38
Notepad	129.73	38.90	31	199	26

5.4.2 Time on Task

The average time on task was 24.63 minutes (Table 10). A one-way ANOVA showed a significant difference ($p < 0.01$) between the average performance times amongst the three sections (Table 11). The ANOVA test was followed by T-tests to determine whether specific differences existed amongst the sections. The results from the T-tests showed a significant difference between the IDLE and PyScripter groups ($p < 0.05$), the IDLE and Notepad groups ($p < 0.01$), and the PyScripter and Notepad groups ($p < 0.01$). This indicated that students who used PyScripter finished their required task quicker than the students using IDLE and Notepad respectively. At the same time, students who used IDLE completed their task quicker than the students using Notepad.

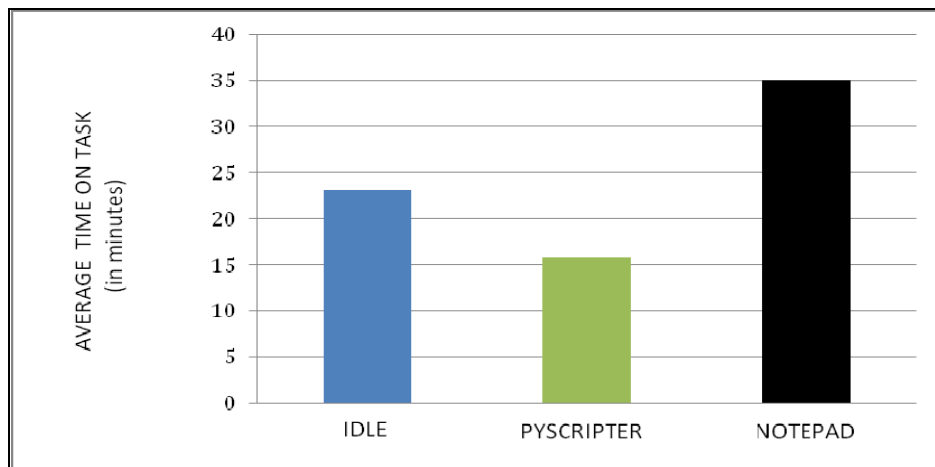


Figure 26: CS160 – Time on Task

Table 10: Time on Task Descriptive Data for CS160 (N = 91)

Average Time	StdDev	Min Time	Max Time
24.63 minutes	13.45	< 1minute	60 minutes

Table 11: Time on Task Descriptive Data Amongst The Three Sections (time in minutes)

Section/Environment	Average Time	StdDev	Min Time	Max Time	N
1 - IDLE	23.05	12.62	4	50	21
2 - PyScripter	15.88	10.89	<1	46	40
3 - Notepad	34.97	16.83	7	60	30

5.4.2.1 Observations while Measuring *Time on Task*

Observations were written based on feedback from the students while programming. Most of their feedback was in the form of questions. For students in the IDLE group, using the Python language was their primary concern. Students in the PyScripter group had similar concerns, but were much less frequent than the IDLE group. Many of the students in the PyScripter group finished their task with no questions or concerns. *Variable misuse* and *improper math operations* were the main concerns seen from these two sections. The Notepad group, on the other hand, had more questions about Notepad than the Python language. Similar to the IDLE and PyScripter groups, variable misuse and improper math operations were common concerns in the Notepad group. However, the majority of their issues related to *command usage*. These issues were seen throughout their programming process. Questions were raised about how to *correctly* create and save a Python file after failed attempts. Later questions were asked about the appropriate commands to interpret their program after failed attempts.

5.4.3 Pennington's Model

5.4.3.1 Version 1 vs. Version 2 (all three groups)

For each group, a T-test was used to determine any significant differences between Versions 1 and 2 in regards of giving the correct answer. Elementary Operations, Control Flow, Data Flow, and Program State showed no significant difference. Program Function however indicated a significant difference; the IDLE group ($p < 0.01$), the PyScripter group ($p < 0.01$) and the Notepad group ($p < 0.01$).

Table 12: Pennington's Model Version 1 vs. Version 2

Group (Version 1 or 2)	N	Elementary Operations	Control Flow	Data Flow	Program State	*Program Function
IDLE (version 1)	22	Correct: 86% Incorrect: 14%	Correct: 95% Incorrect: 5%	Correct: 95% Incorrect: 5%	Correct: 91% Incorrect: 9%	Correct: 73% Incorrect: 27%
IDLE (version 2)	12	Correct: 92% Incorrect: 8%	Correct: 100% Incorrect: 0%	Correct: 92% Incorrect: 8%	Correct: 92% Incorrect: 8%	Correct: 25% Incorrect: 75%
PyScripter (version 1)	17	Correct: 94% Incorrect: 6%	Correct: 88% Incorrect: 12%	Correct: 100% Incorrect: 0%	Correct: 88% Incorrect: 12%	Correct: 94% Incorrect: 6%
PyScripter (version 2)	21	Correct: 95% Incorrect: 5%	Correct: 100% Incorrect: 0%	Correct: 95% Incorrect: 5%	Correct: 95% Incorrect: 5%	Correct: 10% Incorrect: 90%
Notepad (version 1)	18	Correct: 83% Incorrect: 17%	Correct: 78% Incorrect: 22%	Correct: 83% Incorrect: 17%	Correct: 89% Incorrect: 11%	Correct: 83% Incorrect: 17%
Notepad (version 2)	13	Correct: 92% Incorrect: 8%	Correct: 69% Incorrect: 31%	Correct: 92% Incorrect: 8%	Correct: 62% Incorrect: 38%	Correct: 15% Incorrect: 85%

Actual percentages are provided in Table 12. There was the possibility that the students who received Version 2 misinterpreted the question about the Program Function (Table 13). This

particular question was modified on the Version 2 survey to say “smallest” rather than “largest” possible denominations.

Table 13: Program Function (Version 2 Modification)

Version 1	Does this program compute how to give change in the largest possible denominations?
Version 2	Does this program compute how to give change in the smallest possible denominations?

5.4.3.2 Group Comparison

A one-way ANOVA was used to determine any significant difference amongst the groups in regards to providing the correct answer for each question. The results indicated a significant difference ($p < 0.01$) for Control Flow from the Version 2 survey. Students in the Notepad group who took Version 2 provided the incorrect answer more frequently than the IDLE and PyScripter groups respectively. The same was true for Program State in this group ($p < 0.05$).

5.4.3.3 Question Comparison (all three groups)

A one-way ANOVA was used to determine any significant difference amongst the five questions in regards to providing the correct answer. From the Version 1 survey, the results indicated no significant difference. From the Version 2 survey, the results indicated a significant difference ($p < 0.01$) perhaps due to the possible misinterpretation of the Program Function question as previously mentioned in Table 7.

5.5 Environment's Usability Survey

This survey was composed of several attributes to measure the environments' usability: *Initial Impression of the Environment*, *Comfort with Environment*, *Confident with Doing Another Assignment with the Environment*, *Like the Environment*, *Easiest Attributes about the Environment*, and *Hardest Attributes about the Environment*. The responses to the questions in the survey were either multiple choice or open-ended. The following subsections detail the measurements of these areas and the results. A summary of these results is provided in Tables 14 – 15b.

5.5.1 Initial Impression about the Environment

This question was open-ended. The responses were quantified into three categories: *positive*, *non-positive*, and *no response*. Non-positive responses consist of either neutral/confused or negative feelings about the environment. A one-way ANOVA indicated a significant difference ($p < 0.01$). Afterwards, T-tests indicated a significant difference for each T-test: IDLE vs. PyScripter ($p = 0.05$), IDLE vs. Notepad ($p = 0.05$), PyScripter vs. Notepad ($p < 0.01$). These results showed that the Notepad group had a less positive initial impression than the IDLE and PyScripter groups respectively. In addition, students in the IDLE group had a less positive initial impression than the PyScripter group.

5.5.2 Comfort with Environment

The answers for this question were based on a 7-point Likert scale, ranging from *not comfortable at all* to *absolutely comfortable*. A one-way ANOVA indicated a significant difference ($p < 0.01$). Afterwards, T-tests indicated a significant difference for two of the pairings: IDLE vs. PyScripter ($p < 0.01$) and IDLE vs. Notepad ($p < 0.05$). These results showed that the IDLE group was less comfortable with using IDLE than the PyScripter group with PyScripter and the Notepad group with Notepad respectively. The PyScripter and Notepad groups showed no significant difference between each other.

5.5.3 Confidence with Doing Another Assignment with the Environment

The answers for this question were based on a 7-point Likert scale, ranging from *not confident at all* to *absolutely confident*. A one-way ANOVA indicated a significant difference ($p < 0.01$). Afterwards, T-tests indicated a significant difference for two of the pairings: IDLE vs. PyScripter ($p < 0.01$) and IDLE vs. Notepad ($p < 0.05$). These results showed that the IDLE group was less confident with using IDLE to do another assignment than the PyScripter group with PyScripter and the Notepad group with Notepad respectively. The PyScripter and Notepad groups showed no significant difference between each other.

5.5.4 Like the Environment

The answers for this question were based on a 7-point Likert scale, ranging from *not at all* to *absolutely like*. A one-way ANOVA indicated a significant difference ($p < 0.01$). Afterwards, T-tests indicated a significant difference for two of the pairings: IDLE vs. PyScripter ($p < 0.01$) and PyScripter vs. Notepad ($p < 0.01$). The results showed that the students in the IDLE

and Notepad groups liked IDLE and Notepad respectively less than the PyScripter group with PyScripter. The IDLE and Notepad groups showed no significant difference between each other.

5.5.5 Easiest Attributes about the Environment

This question was open-ended with the responses quantified into five categories: *Python Attributes*, *Environment Attributes*, *Familiarity*, *Nothing/No Response* and *I Don't Know*. Python Attributes represented students who gave a response about the Python language. Environment Attributes represented students who gave a response about their respective environment based on its features. Familiarity represented students who responded based on a previous experience with programming. The categories of *Nothing/No Response* and *I Don't Know* represented students who actually provided such responses.

A one-way ANOVA indicated no significant difference amongst the three groups. Since many of the students in CS 160 were not exposed to Python prior to this study, several of them responded more frequently about the easiest attributes of the Python language itself rather than their respective environment. A T-test indicated a significant difference ($p < 0.05$) between responses towards the Python language and the respective environments. Additional T-tests were used to determine any significant differences within each group. The results indicated a significant difference ($p < 0.01$) for only the IDLE group. These results showed that the IDLE group responded more frequently about the easy attributes of the Python language rather than the IDLE environment. The frequency of responses to *Familiarity*, *Nothing/No Response*, and *I Don't Know* were insignificant.

5.5.6 Hardest Attributes about the Environment

This question was also open-ended with the same response categories used in the easiest attributes. A one-way ANOVA indicated a significant difference ($p < 0.01$). Afterwards, T-tests indicated a significant difference for two of the pairings: IDLE vs. Notepad ($p < 0.01$) and PyScripter vs. Notepad ($p < 0.01$). These results showed that Notepad received more responses concerning its hard attributes than IDLE and PyScripter respectively.

In regards to the Python language itself, a one-way ANOVA indicated a slight significant difference ($p = 0.054$). Afterwards, T-tests indicated a significant difference for two of the pairings: the IDLE group vs. the Notepad group ($p = 0.01$) and the PyScripter group vs. the Notepad group ($p < 0.05$). These results showed that students in the Notepad group gave fewer responses about the hardest attributes of the Python language than the IDLE and PyScripter groups respectively. The frequency of responses to *Familiarity*, *Nothing/No Response*, and *I Don't Know* were insignificant.

5.5.7 Experiences with Other Environments (Besides PREOP)

This question was open-ended. A one-way ANOVA was used to determine if certain sections had more prior experience with other environments besides PREOP. The results indicated a significant difference ($p < 0.01$). Afterwards, T-tests were used to compare each group against another. The results indicated a significant difference for two of the T-tests: the IDLE group vs. the PyScripter group ($p < 0.01$) and the IDLE group vs. the Notepad group ($p < 0.01$). These results showed that the IDLE group has less experience with using other environments (besides PREOP) than the PyScripter and Notepad groups respectively.

A one-way ANOVA was also used to determine if these other environments were mandatory for another course. The results indicated a significant difference ($p < 0.05$). Afterwards, T-test indicated a significant difference for only one of the pairings: the IDLE group vs. the PyScripter group ($p < 0.01$). These results not only showed that the PyScripter group had more experience with other environments than the IDLE group, but also that they were mandatory for another course. The PyScripter and Notepad groups showed no significant difference between them.

An additional T-test was used for the PyScripter group to determine whether their experience with other environments were actually IDEs. For the PyScripter group, the results were significant ($p < 0.01$). These results showed that most of these students (68%) had prior experience with IDEs. As previously mentioned, many of the students in the PyScripter group were ECE majors. Traditionally at this university, all ECE majors must take CS285, which teaches the C language using the CodeBlocks IDE. Similar to PyScripter, CodeBlocks is a rich-featured IDE. Out of the 68% of these students who had prior exposure to IDEs, 90% of them had experience with CodeBlocks.

Table 14: CS 160 Environment Usability Data

Student Representation (N=102*)		
Initial Impression	Comfort with Environment	Confident with Doing Another Assignment
Positive - 37% Non-Positive - 55% No Response - 8%	Not Comfortable At All - 7% Mostly Not Comfortable - 12% Slightly Comfortable - 18% 50/50 - 18% Fairly Comfortable - 24% Mostly Comfortable - 16% Absolutely Comfortable - 7%	Not Confident At All - 10% Mostly Not Confident - 9% Slightly Confident - 19% 50/50 - 20% Fairly Confident - 17% Mostly Confident - 13% Absolutely Confident - 14%
Like the Environment	Easiest Attributes	Hardest Attributes
Not At All - 13% Mostly Do Not Like - 10% Slightly Like - 11% 50/50 - 25% Fairly Like - 20% Mostly Like - 10% Absolutely Like - 12%	Writing The Code/Python Attributes - 44% Environment Attributes - 30% Familiarity - 7% No Response/Nothing - 14% I Don't Know - 5%	Writing The Code/Python Attributes - 48% Environment Attributes - 18% Familiarity - 12% No Response/Nothing - 22% I Don't Know - 1%
Prior Experience with other Environments (besides PREOP)	Environment Mandatory for a Course	Prior Experience with Visual or Command Line Environments
Yes – 48% No – 52% <i>*Two students did not provide a response</i>	Yes – 35% No – 65% <i>*Two students did not provide a response.</i>	Student Representation (N=47**) Visual – 43% Command Line – 57% <i>* One student had experience with both.</i>
<i>*Number of responses after Time on Task was conducted;</i> <i>**Number of responses with prior programming experience after Time on Task was conducted.</i>		

Table 15a: Section-by-Section Environment Usability Data

Group	Initial Impression	Comfort with Environment	Confident with Doing Another Assignment
<p>IDLE (N=34*)</p>	<p>Positive - 35% Non-Positive - 53% No Response - 12%</p>	<p>Not Comfortable At All - 12% Mostly Not Comfortable - 53% Slightly Comfortable - 12% 50/50 - 18% Fairly Comfortable - 21% Mostly Comfortable - 9%</p> <p><i>* No student indicated Absolutely Comfortable.</i></p>	<p>Not Confident At All - 18% Mostly Not Confident - 15% Slightly Confident - 18% 50/50 - 26% Fairly Confident - 12% Mostly Confident - 9% Absolutely Confident - 3%</p>
<p>PyScripter (N=38*)</p>	<p>Positive - 55% Non-Positive - 45% No Response - 0%</p>	<p>Mostly Not Comfortable - 11% Slightly Comfortable - 18% 50/50 - 11% Fairly Comfortable - 29% Mostly Comfortable - 21% Absolutely Comfortable - 11%</p> <p><i>* No student indicated Not Comfortable At All.</i></p>	<p>Not Confident At All - 5% Slightly Confident - 21% 50/50 - 13% Fairly Confident - 29% Mostly Confident - 13% Absolutely Confident - 18%</p> <p><i>* No student indicated Mostly Not Confident.</i></p>
<p>Notepad (N=30*)</p>	<p>Positive - 17% Non-Positive - 70% No Response - 13%</p>	<p>Not Comfortable At All - 10% Mostly Not Comfortable - 7% Slightly Comfortable - 10% 50/50 - 27% Fairly Comfortable - 20% Mostly Comfortable - 17% Absolutely Comfortable - 10%</p>	<p>Not Confident At All - 7% Mostly Not Confident - 13% Slightly Confident - 17% 50/50 - 20% Fairly Confident - 7% Mostly Confident - 17% Absolutely Confident - 20%</p>
<p><i>*Number of responses after Time on Task was conducted.</i></p>			

Table 15b: Section-by-Section Environment Usability Data (CONT'D)

Group	Like the Environment	Easiest Attributes	Hardest Attributes
<p>IDLE (N=34*)</p>	<p>Not At All - 18% Mostly Do Not Like - 21% Slightly Like - 6% 50/50 - 30% Fairly Like - 15% Mostly Like - 9% Absolutely Like - 3%</p>	<p>Writing The Code/Python Attributes - 50% Environment Attributes - 18% Familiarity - 6% No Response/Nothing - 18% I Don't Know - 9%</p>	<p>Writing The Code/Python Attributes - 59% Environment Attributes - 6% Familiarity - 15% No Response/Nothing - 21%</p> <p><i>* No student indicated I Don't Know.</i></p>
<p>PyScripter (N=38*)</p>	<p>Not At All - 5% Slightly Like - 16% 50/50 - 21% Fairly Like - 18% Mostly Like - 18% Absolutely Like - 21%</p> <p><i>* No student indicated Mostly Do Not Like.</i></p>	<p>Writing The Code/Python Attributes - 47% Environment Attributes - 37% Familiarity - 3% No Response/Nothing - 11% I Don't Know - 3%</p>	<p>Writing The Code/Python Attributes - 53% Environment Attributes - 11% Familiarity - 11% No Response/Nothing - 24% I Don't Know - 3%</p>
<p>Notepad (N=30*)</p>	<p>Not At All - 17% Mostly Do Not Like - 10% Slightly Like - 10% 50/50 - 27% Fairly Like - 27% Absolutely Like - 10%</p> <p><i>* No student indicated Mostly Like.</i></p>	<p>Writing The Code/Python Attributes - 33% Environment Attributes - 37% Familiarity - 13% No Response/Nothing - 13% I Don't Know - 3%</p>	<p>Writing The Code/Python Attributes - 30% Environment Attributes - 40% Familiarity - 10% No Response/Nothing - 20%</p> <p><i>* No student indicated I Don't Know.</i></p>

**Number of responses after Time on Task was conducted.*

5.6 Discussion

The IDLE group had less prior programming experience than their counterparts in the PyScripter and Notepad groups. This factor may have impacted a majority of the results seen from this group. They were found to be less confident in their programming abilities, less comfortable with IDLE after using it, and less confident about doing another assignment. They also did not like IDLE as much as students who liked PyScripter. Their lack of programming experience was obvious when asked about the ease or difficulty of using IDLE. Instead of providing positive responses about IDLE, they expressed comfort about the Python language. Despite lacking programming experience, the IDLE group completed their task significantly faster than the Notepad group.

Students in the PyScripter and Notepad groups showed no differences in their programming experience. They also showed no differences in their comfort with their respective environments as well as their confidence of doing another assignment. However, the PyScripter group had a more positive initial impression, more of a fondness with PyScripter, and a faster completion time than the students using Notepad. Students in the Notepad group (not significantly) had more prior exposure to command line programming through CS150. However, they frequently showed difficulties with using Notepad, which influenced their time to complete the required exercise. In contrast, students using PyScripter rarely demonstrated difficulties about using PyScripter, and a majority of them had prior exposure to IDEs. In addition, 45% of the PyScripter group had a non-positive initial impression. On the other hand, 70% of the Notepad group had a non-positive initial impression. Fifty-three percent of the IDLE group

showed a non-positive impression. However, many of these students did not have prior programming experience unlike the other groups.

Pennington's Model was used to measure the students' understanding of programming. Other than the probable misinterpretation of the question regarding Program Function, the Notepad group students who completed Version 2 of the survey gave more incorrect responses about Control Flow and Program State respectively. Even though they showed more confidence in their programming abilities than the IDLE group, the Notepad group provided fewer correct answers on Pennington's Model. The PyScripter group students showed no notable issues regarding Pennington's Model.

5.7 Summary

In this study, the objective was to measure any difference in impact between moderate and low assistive environments on novice programmers. It is the conclusion that environments with more assistive features can potentially provide a lower learning curve for novice programmers than less assistive features. This study showed that students using IDLE and PyScripter were more efficient with their task than students using Notepad, while PyScripter was also more efficient than IDLE. In terms of usability, IDLE was feasible to use even though the majority of the students in the IDLE group lacked prior programming experience. In the PyScripter group, PyScripter was also feasible to use. However, many of these students also had prior experience with IDEs. A good portion of students in the Notepad group had prior experience with command line programming, but struggled with using Notepad as a whole.

The presence of assistive features within some of these environments is a factor. The IDLE and PyScripter groups used environments that utilize syntax and error highlighting unlike the Notepad group. These particular groups could save and execute their program respectively with a single button click. The Notepad group on the other hand had to use the appropriate commands to perform the same procedure, which was found to be a challenge.

The learning curve for these environments is also a factor. Many of the PyScripter group students had prior exposure to IDEs while a good portion of the Notepad group's students had exposure to a command line environment. However, the PyScripter group outperformed the Notepad group on the required task while the Notepad group commented more about the difficulties of using Notepad. It is possible that students in the PyScripter group acquired a

sufficient amount of understanding from their previous exposure to IDE(s) in order to effectively use PyScripter. On the other hand, command line environments like Notepad may require more time for novices to understand. Another consideration is that the majority of students in the Notepad group who had prior command line programming used Linux and not Windows. When using command line environments, the style of command usage can change from one environment to another as well as from one platform to another. There are also cases where command line environments have their own special set of commands. For example, text editors like Vi/Vim, Emacs, and Pico each have individual command sets. In the case of novices, learning a new set of commands for programming may be confusing to understand at first. In addition, learning different commands for a new environment may increase the learning curve for a novice.

6. CS1 STUDY - PYTHON PROGRAMMING

Chapters 4 and 5 provided detailed results for evaluating programming environments and their effect on novices. However, each study was short-term; Aliceville outreach (5 weeks) and CS1-Laboratory study (1 day). This chapter shows a semester-long study (Fall 2011) that was conducted on participants taking a CS1 course (CS150) involving visual and command line programming. As previously mentioned, CS150 traditionally teaches Python using the VIM command line environment on the Linux platform.

During the Fall 2011 semester, CS150 taught Python using either VIM or IDLE on the Linux platform. Four sections of CS150 were offered (including an honor section). During the latter portion of the semester, an environment switch occurred in order to study the students' acquired mental models at this point. Each section is classified as A, B, or C.

¹Theoretically, Section A would begin the semester with IDLE and use VIM after the scheduled switch. Section B is composed of two classes that would begin the semester with VIM and use IDLE after the scheduled switch. Section C represents the honor section. These students were given the option of using IDLE or VIM based on their preference. During the time of the environment switch, these students would begin using the “other” environment respectively.

¹ There were students who chose not to use their assigned environments due to personal preference, past experiences, etc. This will be discussed later in the chapter (Sections 6.4 & 6.5).

6.1 Environments/Experiment Conditions

The IDLE environment used in this study was a newer version (*version 3.2*) that supported Python 3 (Figure 27). The features and structural make up of this version of IDLE were closely related to the one discussed in Chapter 5 (*version 2.6*). Figure 28 provides a screenshot of the version of VIM (*version 7.3.35*) that was used during this semester.

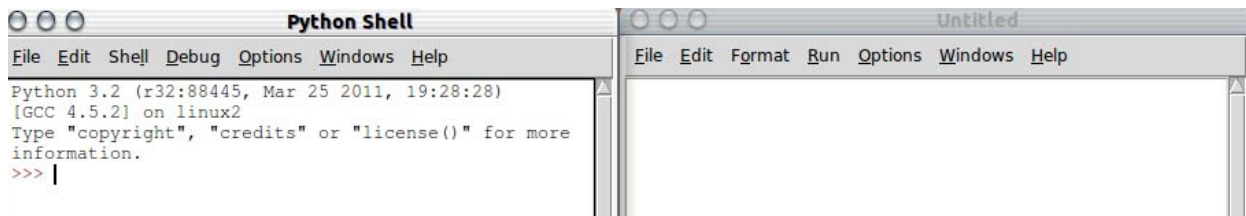


Figure 27: IDLE version 3.2 – Linux platform

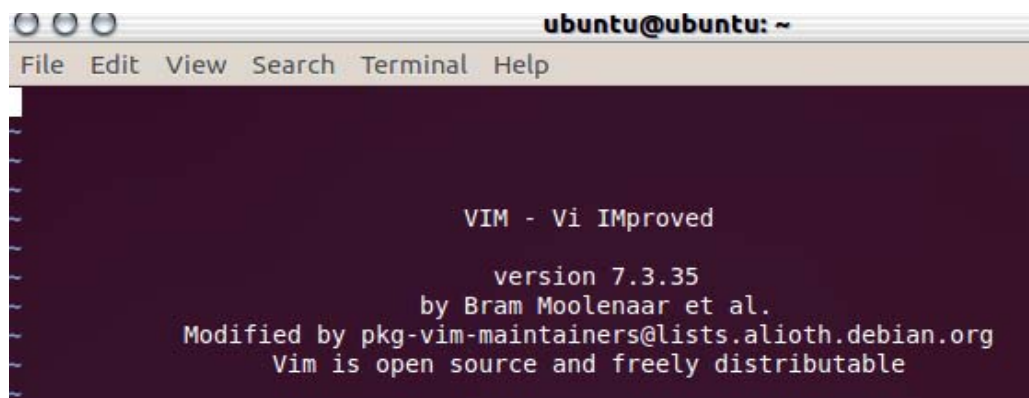


Figure 28: VIM version 7.3.35

Identical measures from the CS1-Laboratory study were used during this study: *a Computer Programming Self-Efficacy Scale, a Pennington’s model survey, Time on Task, and a usability survey*. Since the duration of this study was equivalent to an entire semester, each measure was applied multiple times. A Computer Programming Self-Efficacy Scale and usability survey were administered three times. A Pennington’s model survey was given twice. Time on task was measured during each exam (including final). An additional measure included a protocol analysis that was conducted after the environment switch to measure the students’ acquired mental models from using their original environment. An additional survey for measuring the students’ ability to understanding programming procedures was given twice during the semester. Table 16 provides an outline of the measures applied during this study. The following sections detail each measure.

Table 16: CS1 Study Outline

Month	Tasks: Date (#of students)
September	<ul style="list-style-type: none"> • Exam 0 – Time on Task: September 15 (all students) • Pre-Self-Efficacy/Usability Survey: September 21 (all students)
October	<ul style="list-style-type: none"> • Exam 1 – Time on Task: October 20 (all students) • Self-Efficacy/Usability Survey: October 31 (all students) • Pennington’s Model/Programming Procedures: October 31 (all students)
November	<ul style="list-style-type: none"> • Switch Environments: November 2 – 7 (all students) • Audio/Video – Protocol Analysis: November 10 (4 – 8 students) • Exam 2 – Time on Task: November 17 (all students)
December	<ul style="list-style-type: none"> • Self-Efficacy/Usability Survey: December 5 (all students) • Pennington’s Model/Programming Procedures: December 5 (all students) • Final Exam – Time on Task: December 15 (all students)

6.2 Demographics

The student representation for CS150 was assessed three times throughout the semester while measuring their programming self-efficacy and usability of IDLE or VIM. These assessments were categorized as *First Survey*, *Second Survey*, and *Third Survey*. The following tables (Tables 17a - 22d) represent the demographics for CS150 as an entire population as well as section-by-section for each assessment. The second and third surveys only represent a portion of the variables measured in the first survey, which include *programming skills*, *computer knowledge*, *program intimidation*, *computer science intimidation*, *grade expectation in CS150*, and *gender*. The demographic statistics for each assessment were calculated through one-way ANOVAs and T-Tests. In addition, a Bernoulli's test was applied to samples with a low representation in order to control any potential Type I or Type II Errors. Each assessment is discussed in detail as a subsection.

6.2.1 Demographics: First Survey

The demographics of the First Survey represented a population of 119 students (Tables 17a - 18c). Sixteen variables were used to obtain feedback from the students, these include: *major*, *classification*, *GPA*, *gender*, *programming experience*, *program intimidation (as a whole)*, *programming intimidation (males)*, *programming intimidation (females)*, *computer science intimidation (as a whole)*, *computer science intimidation (males)*, *computer science intimidation (females)*, *grade expected in CS150*, *programming skills*, *computer knowledge*, *another programming class requirement*, and *another programming class to graduate*.

The student representation for CS150 consisted of different majors and classification levels. Section A had significantly more Computer Science majors than Sections B ($p < 0.01$) and C ($p < 0.01$). Section C had a significantly higher freshman enrollment than Section B ($p < 0.05$). This particular statistic may show why Section C also had a significantly higher percentage of first semester students with no current GPA than sections A ($p < 0.05$) and B ($p < 0.01$). However, students in Section C expected to earn a higher letter grade in CS150 than their counterparts in Section B ($p < 0.01$) at this point in the semester. Each section had a higher percentage of male students. Each section also showed a majority of students who lacked prior programming experience coming into CS150. When observing program intimidation and computer science intimidation, each section showed a higher percentage of students who were not intimidated. A majority of students in sections A and B also rated their programming skills to be *average* in comparison to their peers. However, a majority of students in section C believed to have *somewhat more* programming skills than their peers. All three sections showed a majority of students to report an *average* knowledge about computers in comparison to their peers. In addition, a majority needed to take another programming course during the following semester.

6.2.2 Demographics: Second Survey

The demographics of the Second Survey also represented a population of 119 students (Tables 19 - 20b). A majority of students from each section had *average* programming skills and computer knowledge. Section C had a significantly lower amount of students who were intimidated by programming at this point than sections A ($p < 0.01$) and B ($p < 0.01$). This section also showed a significantly lower number of students who were intimidated by computer science at this point than sections A ($p < 0.01$) and B ($p < 0.01$). Each section showed a majority of

students who expected to earn a grade of A^- , A , or A^+ in CS150. These sections also showed a higher male percentage at this point in the semester.

6.2.3 Demographics: Third Survey

The demographics of the Third Survey represented a population of 126 students (Tables 21 - 22b), including a duplicate representation of Section C students who received two surveys during this assessment. In this subsection, section C students are divided into two subsections: *Section C-IDLE* and *Section C-VIM*. Section A only had 17 respondents at this point of the semester. Therefore, a Bernoulli test was applied to each statistical analysis involving Section A's sample.

Each section reported a majority to have *average* programming skills and computer knowledge. Sections C-IDLE ($p=0.01$) and C-VIM ($p<0.05$) respectively had a significantly lower number of students who were intimidated by programming at this point than section B. Sections C-IDLE ($p<0.05$) and C-VIM ($p<0.05$) also showed a significantly lower number of students who were intimidated by computer science at this point than section B. Each section reported a majority of students who expected to earn a grade of A^- , A , or A^+ in CS150. These sections also showed a higher male percentage at this point in the semester.

Table 17a: CS 150 Demographics – First Survey

Student Representation (N=119)			
Major	Classification	Current GPA	Gender
Computer Science - 61% Electrical Engineering - 3% Computer Engineering - 3% MIS - 1% Math - 6% Other - 22% Double Major (including CS) - 1% Double Major (excluding CS) - 3%	Freshmen - 40% Sophomore - 32% Junior - 19% Senior - 8% Other - 3% <i>*one student did not provide an answer</i>	3.0 - 4.0: 47% 2.0 - 3.0: 17% 1.0 - 2.0: 4% <1.0: 0% First Semester (No GPA): 31% <i>*one student did not provide an answer</i>	Male - 73% Female - 27% <i>*two students did not provide an answer</i>
Programming Experience	Intimidated by Programming	Males Intimidated by Programming (N=85)	Females Intimidated by Programming (N=32)
High School programming - 16% Another College Course - 16% No Prior Experience - 68% <i>*three students did not provide an answer</i>	Yes - 36% No - 64% <i>*one student did not provide an answer</i>	Yes - 32% No - 68%	Yes - 50% No - 50%

Table 17b: CS 150 Demographics – First Survey (CONT'D)

Student Representation (N=119)			
Intimidated by Computer Science	Males Intimidated by Computer Science (N=85)	Females Intimidated by Computer Science (N=32)	Grade Expected in CS 150
<p>Yes - 24% No - 76%</p> <p><i>*one student did not provide an answer</i></p>	<p>Yes - 20% No - 80%</p>	<p>Yes - 34% No - 66%</p>	<p>A+, A, A- : 66% B+, B, B- : 19% C+, C, C- : 12% D+, D, D- : 1% F : 0% Not Taking CS150 for a grade: 1%</p> <p><i>*two students did not provide an answer</i></p>
Programming Skills	Computer Knowledge	Taking Another Programming Class	Need Another Programming Class to Graduate
<p>I have a lot more skill - 10% I have somewhat more skill - 26% I have average skill - 37% I have somewhat less skill - 15% I have a lot less skill - 11%</p> <p><i>*one student did not provide an answer</i></p>	<p>I have a lot more knowledge - 11% I have somewhat more knowledge - 24% I have average knowledge - 49% I have somewhat less knowledge - 12% I have a lot less knowledge - 4%</p> <p><i>*one student did not provide an answer</i></p>	<p>Next semester: 84% Another semester: 7% Never: 9%</p> <p><i>*two students did not provide an answer</i></p>	<p>Yes - 84% No - 16% I Don't Know - 1%</p> <p><i>*two students did not provide an answer</i></p>

Table 18a: Section-by-Section Demographics – First Survey

Group	Major	Classification	GPA	Gender
Section A (N=33)	Computer Science - 85% Electrical Engineering - 0% Computer Engineering - 0% MIS - 0% Math - 6% Other - 9% Double Major (including CS) - 0% Double Major (excluding CS) - 0%	Freshmen - 34% Sophomore - 42% Junior - 15% Senior - 9% Other - 0%	3.0 - 4.0: 58% 2.0 - 3.0: 24% 1.0 - 2.0: 0% <1.0: 0% First Semester (No GPA): 18%	Male - 73% Female - 27%
Section B* (N=46)	Computer Science - 49% Electrical Engineering - 2% Computer Engineering - 0% MIS - 2% Math - 9% Other - 29% Double Major (including CS) - 2% Double Major (excluding CS) - 7% <i>*one student did not provide an answer</i>	Freshmen - 31% Sophomore - 27% Junior - 29% Senior - 11% Other - 2% <i>*one student did not provide an answer</i>	3.0 - 4.0: 51% 2.0 - 3.0: 22% 1.0 - 2.0: 0% <1.0: 0% First Semester (No GPA): 27%	Male - 74% Female - 26%
Section C (N=40)	Computer Science - 56% Electrical Engineering - 5% Computer Engineering - 7% MIS - 0% Math - 5% Other - 27% Double Major (including CS) - 0% Double Major (excluding CS) - 3%	Freshmen - 55% Sophomore - 28% Junior - 10% Senior - 3% Other - 5%	3.0 - 4.0: 35% 2.0 - 3.0: 5% 1.0 - 2.0: 13% <1.0: 0% First Semester (No GPA): 48%	Male - 85% Female - 15%

Statistical Significance

Major: A one-way ANOVA was conducted ($p < 0.01$); T-tests showed a significant difference between Sections A and B ($p < 0.01$) and Sections A and C ($p < 0.01$).

Classification: A one-way ANOVA was conducted ($p = 0.05$); A t-test showed a significant difference between Sections B and C ($p < 0.05$).

No GPA (First Semester): A one-way ANOVA was conducted ($p < 0.05$); T-tests showed a significant difference between Sections A and C ($p < 0.05$) and Sections B and C ($p < 0.01$).

**Indicates two sections.*

Table 18b: Section-by-Section Demographics – First Survey (CONT'D)

Group	Grade Expected in CS150	Programming Experience	Intimidated By Programming	Intimidated By Computer Science
Section A (N=33)	A+, A, A- : 66% B+, B, B- : 18% C+, C, C- : 16% D+, D, D- : 0% F : 0% Not Taking CS150 for a grade: 1%	High School programming - 9% Another College Course - 25% No Prior Experience - 66%	Yes - 39% No - 61%	Yes - 27% No - 73%
Section B* (N=46)	A+, A, A- : 56% B+, B, B- : 31% C+, C, C- : 11% D+, D, D- : 0% F : 0% Not Taking CS150 for a grade: 2% <i>*one student did not provide an answer</i>	High School programming - 11% Another College Course - 9% No Prior Experience - 80%	Yes - 44% No - 56%	Yes - 31% No - 69%
Section C (N=40)	A+, A, A- : 83% B+, B, B- : 8% C+, C, C- : 9% D+, D, D- : 0% F : 0% Not Taking CS150 for a grade: 0%	High School programming - 25% Another College Course - 17% No Prior Experience - 58%	Yes - 25% No - 75%	Yes - 13% No - 87%
Statistical Significance				
Graded Expected in CS150 (A-, A, A+): A one-way ANOVA was conducted ($p < 0.05$); A t-test showed a significant difference between Sections B and C ($p < 0.01$).				
Graded Expected in CS150 (B-, B, B+): A one-way ANOVA was conducted ($p < 0.05$); A t-test showed a significant difference between Sections B and C ($p < 0.01$).				
<i>*Indicates two sections.</i>				

Table 18c: Section-by-Section Demographics – First Survey (CONT'D)

Group	Programming Skills (in comparison to others)	Computer Knowledge (in comparison to others)	Taking Another Programming Class	Need Another Programming Class to Graduate
Section A (N=33)	I have a lot more skill - 6% I have somewhat more skill - 27% I have average skill - 33% I have somewhat less skill - 12% I have a lot less skill - 21%	I have a lot more knowledge - 21% I have somewhat more knowledge - 12% I have average knowledge - 45% I have somewhat less knowledge - 15% I have a lot less knowledge - 6%	Next semester: 84% Another semester: 7% Never: 9% <i>*one student did not provide an answer</i>	Yes - 91% No - 9% I Don't Know - 0% <i>*one student did not provide an answer</i>
Section B* (N=46)	I have a lot more skill - 11% I have somewhat more skill - 20% I have average skill - 47% I have somewhat less skill - 11% I have a lot less skill - 11% <i>*one student did not provide an answer</i>	I have a lot more knowledge - 9% I have somewhat more knowledge - 27% I have average knowledge - 49% I have somewhat less knowledge - 9% I have a lot less knowledge - 7% <i>*one student did not provide an answer</i>	Next semester: 71% Another semester: 11% Never: 18% <i>*one student did not provide an answer</i>	Yes - 79% No - 20% I Don't Know - 1% <i>*two students did not provide an answer</i>
Section C (N=40)	I have a lot more skill - 13% I have somewhat more skill - 33% I have average skill - 30% I have somewhat less skill - 23% I have a lot less skill - 3% <i>*one student did not provide an answer</i>	I have a lot more knowledge - 5% I have somewhat more knowledge - 30% I have average knowledge - 53% I have somewhat less knowledge - 13% I have a lot less knowledge - 0% <i>*one student did not provide an answer</i>	Next semester: 97% Another semester: 3% Never: 0% <i>*two students did not provide an answer</i>	Yes - 85% No - 15% I Don't Know - 0% <i>*two students did not provide an answer</i>
<i>*Indicates two sections.</i>				

Table 19: CS 150 Demographics – Second Survey

Student Representation (N=119)		
Programming Skills (in comparison to others)	Computer Knowledge (in comparison to others)	Intimidated By Programming
<p>I have a lot more skill - 9% I have somewhat more skill - 20% I have average skill - 43% I have somewhat less skill - 22% I have a lot less skill - 6%</p>	<p>I have a lot more knowledge - 11% I have somewhat more knowledge - 23% I have average knowledge - 49% I have somewhat less knowledge - 11% I have a lot less knowledge - 6%</p> <p><i>*one student did not provide an answer</i></p>	<p>Yes - 43% No - 57%</p> <p><i>*one student did not provide an answer</i></p>
Intimidated by Computer Science	Grade Expected in CS150	Gender
<p>Yes - 37% No - 63%</p> <p><i>*two students did not provide an answer</i></p>	<p>A+, A, A- : 55% B+, B, B- : 30% C+, C, C- : 14% D+, D, D- : 2% F : 0% Not Taking CS150 for a grade: 0%</p> <p><i>*two students did not provide an answer</i></p>	<p>Male - 78% Female - 22%</p> <p><i>*one student did not provide an answer</i></p>

Table 20a: Section-by-Section Demographics – Second Survey

Group	Programming Skills (in comparison to others)	Computer Knowledge (in comparison to others)	Intimidated By Programming
Section A (N=33)	I have a lot more skill - 11% I have somewhat more skill - 14% I have average skill - 39% I have somewhat less skill - 29% I have a lot less skill - 7%	I have a lot more knowledge - 18% I have somewhat more knowledge - 18% I have average knowledge - 50% I have somewhat less knowledge - 11% I have a lot less knowledge - 4%	Yes - 41% No - 59% <i>*one student did not provide an answer</i>
Section B* (N=46)	I have a lot more skill - 9% I have somewhat more skill - 17% I have average skill - 42% I have somewhat less skill - 25% I have a lot less skill - 8%	I have a lot more knowledge - 8% I have somewhat more knowledge - 26% I have average knowledge - 43% I have somewhat less knowledge - 11% I have a lot less knowledge - 11%	Yes - 47% No - 53%
Section C (N=40)	I have a lot more skill - 8% I have somewhat more skill - 29% I have average skill - 47% I have somewhat less skill - 13% I have a lot less skill - 3%	I have a lot more knowledge - 11% I have somewhat more knowledge - 22% I have average knowledge - 57% I have somewhat less knowledge - 11% I have a lot less knowledge - 0%	Yes - 18% No - 82% <i>*one student did not provide an answer</i>
Statistical Significance Intimidated By Programming: A one-way ANOVA was conducted ($p < 0.01$); T-tests showed a significant difference between Sections A and C ($p < 0.01$) and Sections B and C ($p < 0.01$).			
<i>*Indicates two sections.</i>			

Table 20b: Section-by-Section Demographics – Second Survey (CONT'D)

Group	Intimidated By Computer Science	Grade Expected in CS150	Gender
<p>Section A (N=33)</p>	<p>Yes - 56% No - 44%</p> <p><i>*one student did not provide an answer</i></p>	<p>A+, A, A- : 52% B+, B, B- : 26% C+, C, C- : 22% D+, D, D- : 0% F : 0% Not Taking CS150 for a grade: 0%</p> <p><i>*one student did not provide an answer</i></p>	<p>Male - 67% Female - 33%</p> <p><i>*one student did not provide an answer</i></p>
<p>Section B* (N=46)</p>	<p>Yes - 46% No - 54%</p> <p><i>*one student did not provide an answer</i></p>	<p>A+, A, A- : 47% B+, B, B- : 36% C+, C, C- : 15% D+, D, D- : 2% F : 0% Not Taking CS150 for a grade: 0%</p> <p><i>*two students did not provide an answer</i></p>	<p>Male - 75% Female - 25%</p>
<p>Section C (N=40)</p>	<p>Yes - 11% No - 89%</p>	<p>A+, A, A- : 71% B+, B, B- : 22% C+, C, C- : 6% D+, D, D- : 0% F : 0% Not Taking CS150 for a grade: 0%</p>	<p>Male - 90% Female - 10%</p>
<p align="center">Statistical Significance</p> <p>Intimidated By Computer Science: A one-way ANOVA was conducted ($p < 0.01$); T-tests showed a significant difference between Sections A and C ($p < 0.01$) and Sections B and C ($p < 0.01$).</p>			
<p align="center"><i>*Indicates two sections.</i></p>			

Table 21: CS 150 Demographics – Third Survey

Student Representation (N=126*)		
Programming Skills (in comparison to others)	Computer Knowledge (in comparison to others)	Intimidated by Programming
<p>I have a lot more skill - 14% I have somewhat more skill - 23% I have average skill - 45% I have somewhat less skill - 11% I have a lot less skill - 6%</p> <p><i>*two students did not provide an answer</i></p>	<p>I have a lot more knowledge - 12% I have somewhat more knowledge - 23% I have average knowledge - 53% I have somewhat less knowledge - 7% I have a lot less knowledge - 4%</p> <p><i>*two students did not provide an answer</i></p>	<p>Yes - 40% No - 60%</p> <p><i>*two students did not provide an answer</i></p>
Intimidated by Computer Science	Grade Expected in CS150	Gender
<p>Yes - 33% No - 67%</p> <p><i>*two students did not provide an answer</i></p>	<p>A+, A, A- : 48% B+, B, B- : 36% C+, C, C- : 10% D+, D, D- : 2% F : 3% Not Taking CS150 for a grade: 0%</p> <p><i>*two students did not provide an answer</i></p>	<p>Male - 76% Female - 24%</p> <p><i>*two students did not provide an answer</i></p>
<p><i>*Section C was surveyed twice; N includes a duplicate representation of Section C.</i></p>		

Table 22a: Section-by-Section Demographics – Third Survey

Group	Programming Skills (in comparison to others)	Computer Knowledge (in comparison to others)	Intimidated by Programming
<p>Section A (N=17)</p>	<p>I have a lot more skill - 18% I have somewhat more skill - 35% I have average skill - 41% I have somewhat less skill - 6% I have a lot less skill - 0%</p>	<p>I have a lot more knowledge - 24% I have somewhat more knowledge - 12% I have average knowledge - 47% I have somewhat less knowledge - 18% I have a lot less knowledge - 0%</p>	<p>Yes - 47% No - 53%</p>
<p>Section B* (N=44)</p>	<p>I have a lot more skill - 11% I have somewhat more skill - 20% I have average skill - 39% I have somewhat less skill - 16% I have a lot less skill - 14%</p>	<p>I have a lot more knowledge - 14% I have somewhat more knowledge - 30% I have average knowledge - 36% I have somewhat less knowledge - 9% I have a lot less knowledge - 11%</p>	<p>Yes - 52% No - 48% <i>*one student did not provide an answer</i></p>
<p>Section C - IDLE (N=33**)</p>	<p>I have a lot more skill - 13% I have somewhat more skill - 25% I have average skill - 47% I have somewhat less skill - 3% I have a lot less skill - 13%</p>	<p>I have a lot more knowledge - 6% I have somewhat more knowledge - 22% I have average knowledge - 69% I have somewhat less knowledge - 3% I have a lot less knowledge - 0%</p>	<p>Yes - 25% No - 75% <i>*one student did not provide an answer</i></p>
<p>Section C - VIM (N=31**)</p>	<p>I have a lot more skill - 16% I have somewhat more skill - 19% I have average skill - 55% I have somewhat less skill - 6% I have a lot less skill - 3%</p>	<p>I have a lot more knowledge - 10% I have somewhat more knowledge - 23% I have average knowledge - 65% I have somewhat less knowledge - 3% I have a lot less knowledge - 0%</p>	<p>Yes - 26% No - 74% <i>*one student did not provide an answer</i></p>
<p align="center">Statistical Significance</p> <p>Intimidated By Programming: A one-way ANOVA was conducted ($p=0.05$); T-tests showed a significant difference between Sections B and C - IDLE ($p=0.01$) and Sections B and C - VIM ($p<0.05$).</p>			
<p align="center"><i>*Indicates two sections.</i></p> <p align="center"><i>**Section C was surveyed twice: once for using IDLE and once for using VIM.</i></p>			

Table 22b: Section-by-Section Demographics – Third Survey (CONT'D)

Group	Intimidated by Computer Science	Grade Expected in CS150	Gender
<p>Section A (N=17)</p>	<p>Yes - 47% No - 53%</p>	<p>A+, A, A- : 53% B+, B, B- : 47% C+, C, C- : 0% D+, D, D- : 0% F : 0% Not Taking CS150 for a grade: 0%</p>	<p>Male - 59% Female - 41%</p>
<p>Section B* (N=44)</p>	<p>Yes - 45% No - 55% <i>*one student did not provide an answer</i></p>	<p>A+, A, A- : 41% B+, B, B- : 27% C+, C, C- : 18% D+, D, D- : 5% F : 9% Not Taking CS150 for a grade: 0% <i>*one student did not provide an answer</i></p>	<p>Male - 68% Female - 32%</p>
<p>Section C - IDLE (N=33**)</p>	<p>Yes - 22% No - 78% <i>*one student did not provide an answer</i></p>	<p>A+, A, A- : 53% B+, B, B- : 38% C+, C, C- : 9% D+, D, D- : 0% F : 0% Not Taking CS150 for a grade: 0% <i>*one student did not provide an answer</i></p>	<p>Male - 84% Female - 16%</p>
<p>Section C - VIM (N=31**)</p>	<p>Yes - 19% No - 81% <i>*one student did not provide an answer</i></p>	<p>A+, A, A- : 52% B+, B, B- : 42% C+, C, C- : 6% D+, D, D- : 0% F : 0% Not Taking CS150 for a grade: 0% <i>*one student did not provide an answer</i></p>	<p>Male - 87% Female - 13%</p>
<p style="text-align: center;">Statistical Significance</p> <p>Intimidated By Computer Science: A one-way ANOVA was conducted ($p < 0.05$); T-tests showed a significant difference between Sections B and C - IDLE ($p < 0.05$) and Sections B and C - VIM ($p < 0.05$).</p>			
<p style="text-align: center;"><i>*Indicates two sections.</i> **Section C was surveyed twice: once for using IDLE and once for using VIM.</p>			

6.2.4 Demographics: Survey Comparison

When measuring the average scores for the *programming skills* component of these surveys (using a 5-point Likert scale), the students in sections A and C-IDLE showed a steady increase from the first survey to the third. Sections B and C-VIM showed a decrease in their scores respectively from the first survey to the second. The overall average scores from the three surveys showed sections C-VIM and C-IDLE to be relatively close with scores of 3.32 and 3.29 respectively. Section B had the lowest overall average with 3.02, while Section A had an overall average of 3.12. See Tables 23a - 23d for more details. Overall, students in each section reported the belief to have *average* programming skills in comparison to others.

The *computer knowledge* component of these surveys showed a similar outcome according to averages (using a 5-point Likert scale). Sections A and C-VIM showed a steady increase from the first survey to the third. Section B's average decreased from the first survey to the second. Section C-IDLE showed a very slight decrease from the second survey to the third. Averaging the mean scores of each survey showed sections A and C-VIM to be relatively close with scores of 3.35 and 3.33 respectively. Section B had the lowest overall average with 3.18, while Section C-IDLE had an overall average of 3.30. See Tables 23a - 23d for more details. Overall, students in each section reported the belief to have *average* computation knowledge in comparison to others.

Programming intimidation was measured by averaging the number of *Yes* and *No* responses from each section. Section A showed an increase in programming intimidation during the second survey, but a decrease during the third survey. Section B also showed an increase in programming intimidation during the second survey and a very slight decrease during the third. Sections C-IDLE and C-VIM showed a decrease in programming intimidation during the second

survey, but a slight increase during the third. Averaging the means scores showed that sections C-IDLE and C-VIM to be identical with a score of 1.77 while sections A and B were relatively close with scores of 1.52 and 1.50 respectively. See Tables 23a - 23d for more details. Overall, Section C (as a whole) tended to be less intimidated by programming than students in sections A and B.

Computer Science intimidation was also measured through averaging the number of *Yes* and *No* responses from each section. Section A showed an increase in computer science intimidation during the second survey, but a decrease during the third. Section B also showed an increase in computer science intimidation during the second survey, but a very slight decrease during the third. Sections C-IDLE and C-VIM respectively showed a very slight decrease in their computer science intimidation during the second survey, but an increase after the third. Averaging the means scores showed that sections C-IDLE and C-VIM were relatively close with scores of 1.85 and 1.86 respectively. Sections A and B were also relatively close with scores of 1.57 and 1.59 respectively. See Tables 23a - 23d for more details. Overall, Section C (as a whole) tended to be less intimidated by computer science than students in sections A and B.

A 5-point Likert scale was used to calculate the average *grades expected in CS150*. Section A showed a decrease in their grade expectation during the second survey, but an increase after the third. Section B showed a steady decrease in their grade expectation from the first survey to the third, which was significant ($p < 0.01$). Sections C-IDLE and C-VIM also showed a steady decrease in their grade expectation from the first survey to the third. Averaging the mean scores showed that sections C-IDLE and C-VIM were relatively close with scores of 3.58 and 3.59 respectively. Section B had the lowest overall average of 3.20, while Section had a score of

3.44. See Tables 23a - 23d for more details. Overall, each section (on average) had a grade expectation of B or better for CS150.

The *gender* representation for each survey was measured by averaging the number of *Male* and *Female* responses (*Male* = 1; *Female* = 2). Section A showed a steady increase in female representation from the first survey to the third. Section B showed a decrease in female representation during the second survey, but an increase during the third. Sections C-IDLE and C-VIM showed a decrease in female representation during the second survey, but an increase during the third. Averaging the mean scores showed that sections A and B were relatively close with scores of 1.34 and 1.32 respectively. Sections C-IDLE and C-VIM were also relatively close with scores of 1.14 and 1.13 respectively. See Tables 23a - 23d for more details. Overall, sections A and B appeared to have a higher female representation during these survey assessments than Section C (as a whole).

Table 23a: CS 150 Demographics – Survey Comparison (Section A)

1st Survey (N=33); 2nd Survey (N=28); 3rd Survey (N=17)		
Programming Skills (average based on a 5 point Likert scale: <i>5 = a lot more skill; 1 = a lot less skill</i>)	Computer Knowledge (average based on a 5 point Likert scale: <i>5 = a lot more knowledge; 1 = a lot less knowledge</i>)	Intimidated by Programming (average: 1 = Yes; 2 = No)
1 st survey - 2.85	1 st survey - 3.27	1 st survey - 1.61
2 nd survey - 2.93	2 nd survey - 3.36	2 nd survey - 1.41
3 rd survey - 3.59	3 rd survey - 3.41	3 rd survey - 1.53
Overall Average: 3.12	Overall Average: 3.35	Overall Average: 1.52
Intimidated by Computer Science	Grade Expected in CS150 (average: A= 4; B=3; C=2; D=1; F= 0)	Gender (average: Male = 1; Female = 2)
1 st survey - 1.73	1 st survey - 3.5	1 st survey - 1.28
2 nd survey - 1.44	2 nd survey - 3.29	2 nd survey - 1.33
3 rd survey - 1.53	3 rd survey - 3.52	3 rd survey - 1.41
Overall Average: 1.57	Overall Average: 3.44	Overall Average: 1.34

Table 23b: CS 150 Demographics – Survey Comparison (Section B*)

1st Survey (N=46); 2nd Survey (N=53); 3rd Survey (N=44)		
Programming Skills (average based on a 5 point Likert scale: <i>5 = a lot more skill; 1 = a lot less skill</i>)	Computer Knowledge (average based on a 5 point Likert scale: <i>5 = a lot more knowledge; 1 = a lot less knowledge</i>)	Intimidated by Programming (average: 1 = Yes; 2 = No)
1 st survey - 3.09 2 nd survey - 2.96 3 rd survey - 3.00 Overall Average: 3.02	1 st survey - 3.22 2 nd survey - 3.08 3 rd survey - 3.25 Overall Average: 3.18	1 st survey - 1.56 2 nd survey - 1.47 3 rd survey - 1.48 Overall Average: 1.50
Intimidated by Computer Science	Grade Expected in CS150 (average: A= 4; B=3; C=2; D=1; F=0)	Gender (average: Male = 1; Female = 2)
1 st survey - 1.69 2 nd survey - 1.54 3 rd survey - 1.55 Overall Average: 1.59	1 st survey - 3.45 2 nd survey - 3.28 3 rd survey - 2.86 Overall Average: 3.20	1 st survey - 1.38 2 nd survey - 1.25 3 rd survey - 1.32 Overall Average: 1.32
Statistical Significance Grade Expected in CS150: A one-way ANOVA was conducted ($p < 0.01$); A T-test showed a significant difference between the 1 st and 3 rd surveys ($p < 0.01$).		
<i>*Indicates two sections.</i>		

Table 23c: CS 150 Demographics – Survey Comparison (Section C - IDLE)

1st Survey (N=40); 2nd Survey (N=38); 3rd Survey (N=33**)		
Programming Skills (average based on a 5 point Likert scale: <i>5 = a lot more skill; 1 = a lot less skill</i>)	Computer Knowledge (average based on a 5 point Likert scale: <i>5 = a lot more knowledge; 1 = a lot less knowledge</i>)	Intimidated by Programming (average: 1 = Yes; 2 = No)
<p>1st survey - 3.30</p> <p>2nd survey - 3.26</p> <p>**3rd survey - 3.31</p> <p>Overall Average: 3.29</p>	<p>1st survey - 3.28</p> <p>2nd survey - 3.32</p> <p>**3rd survey - 3.31</p> <p>Overall Average: 3.30</p>	<p>1st survey - 1.75</p> <p>2nd survey - 1.82</p> <p>**3rd survey - 1.75</p> <p>Overall Average: 1.77</p>
Intimidated by Computer Science	Grade Expected in CS150 (average: A= 4; B=3; C=2; D=1; F=0)	Gender (average: Male = 1; Female = 2)
<p>1st survey - 1.88</p> <p>2nd survey - 1.89</p> <p>**3rd survey - 1.78</p> <p>Overall Average: 1.85</p>	<p>1st survey - 3.73</p> <p>2nd survey - 3.58</p> <p>**3rd survey - 3.44</p> <p>Overall Average: 3.58</p>	<p>1st survey - 1.15</p> <p>2nd survey - 1.11</p> <p>**3rd survey - 1.16</p> <p>Overall Average: 1.14</p>
<p><i>*Indicates two sections.</i></p> <p><i>** Responses are based strictly on using IDLE.</i></p>		

Table 23d: CS 150 Demographics – Survey Comparison (Section C - VIM)

1st Survey (N=40); 2nd Survey (N=38); 3rd Survey (N=31)**		
Programming Skills (average based on a 5 point Likert scale: <i>5 = a lot more skill; 1 = a lot less skill</i>)	Computer Knowledge (average based on a 5 point Likert scale: <i>5 = a lot more knowledge; 1 = a lot less knowledge</i>)	Intimidated by Programming (average: 1 = Yes; 2 = No)
1 st survey - 3.30	1 st survey - 3.28	1 st survey - 1.75
2 nd survey - 3.26	2 nd survey - 3.32	2 nd survey - 1.82
**3 rd survey - 3.39	**3 rd survey - 3.39	**3 rd survey - 1.74
Overall Average: 3.32	Overall Average: 3.33	Overall Average: 1.77
Intimidated by Computer Science	Grade Expected in CS150 (average: A= 4; B=3; C=2; D=1; F=0)	Gender (average: Male = 1; Female = 2)
1 st survey - 1.88	1 st survey - 3.73	1 st survey - 1.15
2 nd survey - 1.89	2 nd survey - 3.58	2 nd survey - 1.11
**3 rd survey - 1.81	**3 rd survey - 3.45	**3 rd survey - 1.13
Overall Average: 1.86	Overall Average: 3.59	Overall Average: 1.13
<p><i>*Indicates two sections.</i></p> <p><i>** Responses are based strictly on using VIM.</i></p>		

6.3 Self-Efficacy

The students' self-efficacy for programming was assessed three times throughout the semester. These assessments are categorized as *Pre-Assessment*, *Second Assessment*, and *Final Assessment*. The following tables (Tables 24 - 27) display descriptive data as well as changes in the students' self-efficacy throughout the semester. Each assessment is discussed in detail as a subsection.

6.3.1 Pre-Assessment

The self-efficacy statistics for the Pre-Assessment represented a population of 120 students. Table 24 presents each section (dividing section C into two subsections) and their respective scores. Due to the low representation of students in Section C, who used IDLE, a Bernoulli test was applied to each statistical analysis involving this sample. The scores were calculated through finding the *mean score*, *standard deviation*, *min score*, and *max score*. The mean score was the primary indicator for determining the students' self-efficacy for programming.

Students in Section C, who used IDLE, had the highest mean score of 169.08. The VIM students in this section had the second highest score of 162.15. Section B scored the lowest with a mean score of 126.55, while Section A had a mean score of 136.85. The mean self-scores amongst these four sections were tested using a one-way ANOVA. The ANOVA showed a significant difference ($p < 0.01$). The ANOVA test was followed by T-tests to determine whether specific differences existed amongst the sections. The results from the T-test showed a significant

difference between sections A and C-IDLE ($p < 0.01$, including a Bernoulli test), sections B and C-IDLE ($p < 0.01$, including a Bernoulli test), sections A and C-VIM ($p < 0.05$), and sections B and C-VIM ($p < 0.01$). There was no significant difference between subsections C-IDLE and C-VIM or sections A and B. This indicated that students in Section C, as a whole, were more confident about their programming abilities than their counterparts in sections A and B at this point in the semester.

Table 24: Pre-Self-Efficacy Descriptive Data (N=120) – All Sections

Section	N	Mean	StdDev	Min Score	Max Score
A	33	136.85	43.03	46	215
B*	47	126.55	32.29	52	190
C - IDLE	13	169.08	27.75	124	215
C - VIM	27	162.15	35.45	39	211
All	120	142.00	39.07	39	215
<p>Statistical Significance</p> <p>A one-way ANOVA was conducted ($p < 0.01$); T-tests showed a significant difference between Sections A and C-IDLE ($p < 0.01$, including a Bernoulli test), Sections B and C-VIM ($p < 0.01$), Sections B and C-IDLE ($p < 0.01$, including a Bernoulli test), and Sections A and C-VIM ($p < 0.05$).</p>					
<p>*indicates two sections.</p>					

6.3.2 Second Assessment

The self-efficacy statistics for the Second Assessment represented a population of 119 students. Table 25 presents each section (dividing section C into two subsections) and their respective scores. When comparing mean scores for this assessment, students in Section C, who used VIM, had the highest mean score of 173.55. Students in Section C, who used IDLE, had the second highest mean score with 157.14. Sections A and B had relatively close means with 141.18

and 141.57 respectively. A one-way ANOVA test showed a significant difference ($p < 0.01$) amongst the four sections. T-tests showed a significant difference between sections A and C-VIM ($p < 0.01$) and sections B and C-VIM ($p < 0.01$). This indicated that students in Section C, who used VIM, were more confident about their programming abilities than their counterparts in Sections A and B. Due to the smaller sample size of Section C-IDLE during this assessment (in comparison to the Pre-Assessment), there was the uncertainty of whether this subsection were less confident about their programming abilities in comparison Section C-VIM (or more confident than sections A and B).

Table 25: Second-Self-Efficacy Descriptive Data (N=119) – All Sections

Section	N	Mean	StdDev	Min Score	Max Score
A	28	141.18	42.05	49	217
B*	53	141.57	34.84	62	212
C - IDLE	7**	157.14	37.39	97	200
C - VIM	31	173.55	27.96	100	214
All	119	142.00	39.07	39	215
Statistical Significance					
A one-way ANOVA was conducted ($p < 0.01$); T-tests showed a significant difference between Sections B and C-VIM ($p < 0.01$) and Sections A and C-VIM ($p < 0.01$).					
*indicates two sections. **uncertainty due to sample size.					

6.3.3 Final Assessment

The self-efficacy statistics for the Final Assessment represented a population of 126 students, including a duplicate representation of Section C. A Bernoulli test was applied to each

statistical analysis involving Section A’s sample (N=17). Table 26 presents each section (dividing section C into two subsections) and their respective scores. When comparing mean scores for this assessment, students in Section C, who used VIM, had the highest mean score of 174.81. Students in Section C, who used IDLE, had the second highest mean score with 165.30. Section B scored the lowest with a mean score of 138.67, while Section A had a mean score of 163.65. A one-way ANOVA test showed a significant difference ($p < 0.01$) amongst the four sections. T-tests showed a significant difference between sections A and B ($p < 0.01$, including a Bernoulli test), sections B and C-IDLE ($p < 0.01$), and sections B and C-VIM ($p < 0.01$). There was no significant difference between subsections C-IDLE and C-VIM, sections A and C-IDLE, or sections A and C-VIM. This indicated that students in sections A, C-IDLE, and C-VIM were more confident about their programming abilities than students in Section B.

Table 26: Final-Self-Efficacy Descriptive Data (N=126) – All Sections**

Section	N	Mean	StdDev	Min Score	Max Score
A	17	163.65	23.14	118	216
B*	45	138.67	41.13	35	215
C - IDLE	33	165.30	41.19	50	216
C - VIM	31	174.81	38.44	62	217
All	119	142.00	39.07	39	215
Statistical Significance					
A one-way ANOVA was conducted ($p < 0.01$); T-tests showed a significant difference between Sections B and C-VIM ($p < 0.01$), Sections B and C-IDLE ($p < 0.01$), and Sections A and B ($p < 0.01$, including a Bernoulli test).					
*indicates two sections. Section C was surveyed twice; N includes a duplicate representation of Section C.					

6.3.4 Change in Self-Efficacy

During the second assessment, each section, with the exception of Section C-IDLE, showed an increase in their self-efficacy for programming. Section C-IDLE showed a decrease in their self-efficacy. However, this subsection had a very small sample size during the second assessment. In contrast, Section B showed a significant increase ($p < 0.05$) in their self-efficacy from the pre-assessment (Table 27). During the final assessment, every section showed an increase in their self-efficacy for programming with the exception of Section B. Section B showed a slight decrease in their self-efficacy since the second assessment while Section A showed a significant increase ($p < 0.01$, including a Bernoulli's test). Overall, sections A and C-VIM showed a steady increase in their self-efficacy during the three assessments. Sections B and C-IDLE however showed a decrease in self-efficacy during the final and second assessment respectively.

Table 27: Changes in Self-Efficacy Descriptive Data– All Sections

Section A (averages)	Section B* (averages)	Section C-IDLE (averages)	Section C-VIM (averages)
Pre-Assessment - 136.85	Pre-Assessment - 126.55	Pre-Assessment - 169.08	Pre-Assessment - 162.15
Second Assessment - 141.18	Second Assessment - 141.57	Second Assessment - 157.14	Second Assessment - 173.55
Final Assessment - 163.65	Final Assessment - 138.67	Final Assessment - 165.30	Final Assessment - 174.81
Statistical Significance A T-test (along with a Bernoulli test) showed a significant difference between the First and Final Assessments ($p < 0.01$).	Statistical Significance A T-test showed a significant difference between the First and Second Assessments ($p < 0.05$).	No Statistical Significance	No Statistical Significance

6.4 Comprehension

Three instruments were used to measure comprehension during this study: a *Pennington's Model survey*, a *protocol analysis*, and a *programming procedure survey*. The objective was to study the students' mental model for programming while measuring any changes that occurred to their program understanding throughout the duration of CS150. A Pennington's model and programming procedure survey were given twice during the semester (*once before the environment switch and once after the switch*). The protocol analysis was given during the week of the environment switch. This procedure required students to "think aloud" about their approach for writing a program. The results from each instrument are detailed in the following subsections.

These surveys were also used to detect actual IDLE and VIM users. As previously mentioned, there were students (particularly sections A and B) who chose to use environments contrary to the ones assigned in their respective sections. As part of these surveys, students were asked to state the current environment they were using before and after switching environments. Table 28 provides a representation of students who used IDLE, VIM, or BOTH during these assessments. The results shown in the *Environment Comparison* sections for Pennington's Model and programming procedures respectively are based on the actual users of IDLE and VIM regardless of their section. A T-test showed a significant difference ($p < 0.01$) in VIM usage between the assessments for Section B concerning environment usage. The second assessment showed that students in Section B were using IDLE significantly more than they did during the first assessment.

Table 28: Percentage of IDLE/VIM Users (First and Second Surveys)

First Survey		
Section	N	IDLE vs. VIM Users
A	26	IDLE - 85% VIM - 12% Both - 0% Other - 3%
B*	58	IDLE - 0% VIM - 100% Both - 0% Other - 0%
C	38	IDLE - 21% VIM - 79% Both - 0% Other - 0%
Second Survey		
Section	N	IDLE vs. VIM Users
A	15	IDLE - 93% VIM - 0% Both - 7% Other - 0%
B*	46	IDLE - 26% VIM - 61% Both - 13% Other - 2%
C	33	IDLE - 33% VIM - 52% Both - 15% Other - 0%
Statistical Significance		
Section B: A T-test showed a significant difference between the First and Second Assessments ($p < 0.01$).		
<i>*Indicates two sections.</i>		

6.4.1 Pennington's Model

Similar to the CS1-Laboratory Study (see Chapter 5), two versions of the Pennington's model survey were issued in order to prevent anyone from copying answers. The questions in Version 1 were identical to the questions from Ramalingam and Wiedenbeck's study [131]. Four questions in Version 2 were modified while the question concerning Program Function remained identical to Version 1. This question remained the same in order to prevent a similar misinterpretation that occurred during the CS1-Laboratory Study. Three comparisons were used to discuss the results from this survey, these include: *Section Comparison*, *Environment Comparison*, and *Version Comparison*. One-way ANOVAs and T-Tests were used for each comparison. The following subsections detail the results of each comparison. (Refer back to *Section 2.5.2.2* for further details about Pennington's Model).

6.4.1.1 Section Comparison

For the first survey, one-way ANOVAs were used to determine any significant differences between each section for providing the correct answer on each question. The ANOVAs from both surveys indicated no significant difference. A T-test however indicated a significant difference between Sections B and C ($p < 0.01$) for Version 2's question about Control Flow. Table 29 shows that 68% of the students in Section B, who completed Version 2 of this survey, answered the Control Flow question correctly, while Section C showed a higher percentage (95%) of the correct response for this question.

For the second survey, one-way ANOVAs were also used to determine any significant differences between each section for providing the correct answer for each question. Version 2 indicated no significant difference. Version 1 indicated a significant difference ($p < 0.05$) for

Program State. However, a Bernoulli test showed that these results may not be truly significant. Table 30 displays the results (in the form of *correct* and *incorrect* percentages) for each question.

6.4.1.2 Environment Comparison

For the first survey, T-tests were used to determine any significant differences between IDLE and VIM users for Version 1 and 2 respectively in regards of giving the correct answer. The results indicated no significant difference for each question. Table 31 details the correct and incorrect percentages for each question as well as the respective category for IDLE and VIM users and their respective version of the survey.

For the second survey, one-way ANOVAs were used to determine any significant differences between the users of IDLE, VIM, or BOTH/OTHER environment for providing the correct answer on each question. An ANOVA indicated a significant difference for Elementary Operations ($p < 0.01$). However, the T-tests showed no significant difference for each question. The difference in sample size between these groups may be the reason for these results. Table 32 details the correct and incorrect percentages for each question as well as the respective category for IDLE, VIM, BOTH/OTHER users and their respective version of the survey.

6.4.1.3 Version Comparison

For the first survey, T-tests were used to determine any significant differences between Versions 1 and 2 (for all sections) for providing the correct answer. Elementary Operations, Data Flow, Program State, and Program Function showed no significant difference. Control Flow however indicated a significance difference ($p < 0.01$). Table 33 shows that 93% of the students

using Version 1 of the survey answered the Control Flow question correctly, while 79% of Version 2's students answer this question correctly.

For the second survey, T-tests were used to determine any significant differences between Versions 1 and 2 (for all sections) in regards of giving the correct answer. Elementary Operations, Control Flow, Data Flow, and Program Function showed no significant difference. Program State however indicated a significance difference ($p=0.01$). Table 34 shows that 90% of the students who took Version 1 of the survey answered the Control Flow question correctly, while 70% of Version 2's students answer this question correctly.

6.4.1.4 First vs. Second Survey Comparison

When checking for any significant differences between both surveys for each question, every T-test showed no significant difference. This was true for the section comparison, environment comparison, and version comparison. Tables 35-37 provide average responses ($1 = Correct$; $0 = Incorrect$) for each comparison.

Table 29: Pennington’s Model: Section Comparison (Version 1 vs. Version 2) – First Survey

Section (Version 1 and 2)	N	Elementary Operations	Control Flow	Data Flow	Program State	Program Function
A (version 1)	14	Correct: 100% Incorrect: 0%	Correct: 86% Incorrect: 14%	Correct: 93% Incorrect: 7%	Correct: 79% Incorrect: 21%	Correct: 79% Incorrect: 21%
B* (version 1)	27	Correct: 96% Incorrect: 4%	Correct: 93% Incorrect: 7%	Correct: 96% Incorrect: 4%	Correct: 81% Incorrect: 19%	Correct: 81% Incorrect: 19%
C (version 1)	18	Correct: 89% Incorrect: 11%	Correct: 100% Incorrect: 0%	Correct: 88% Incorrect: 12%	Correct: 94% Incorrect: 6%	Correct: 76% Incorrect: 24%
A (version 2)	12	Correct: 83% Incorrect: 17%	Correct: 83% Incorrect: 17%	Correct: 83% Incorrect: 17%	Correct: 92% Incorrect: 8%	Correct: 75% Incorrect: 25%
B* (version 2)	31	Correct: 97% Incorrect: 3%	Correct: 68% Incorrect: 32%	Correct: 87% Incorrect: 13%	Correct: 65% Incorrect: 35%	Correct: 84% Incorrect: 16%
C (version 2)	20	Correct: 95% Incorrect: 5%	Correct: 95% Incorrect: 5%	Correct: 100% Incorrect: 0%	Correct: 70% Incorrect: 30%	Correct: 85% Incorrect: 15%
Statistical Significance						
Control Flow – Version 2: A T-test showed a significant difference between Section B and C ($p < 0.01$).						
<i>*Indicates two sections.</i>						

Table 30: Pennington’s Model: Section Comparison (Version 1 vs. Version 2) – Second Survey

Section (Version 1 and 2)	N	Elementary Operations	Control Flow	Data Flow	Program State	Program Function
A (version 1)	9	Correct: 100% Incorrect: 0%	Correct: 89% Incorrect: 11%	Correct: 100% Incorrect: 0%	Correct: 67% Incorrect: 33%	Correct: 100% Incorrect: 0%
B* (version 1)	25	Correct: 94% Incorrect: 6%	Correct: 92% Incorrect: 8%	Correct: 84% Incorrect: 16%	Correct: 92% Incorrect: 8%	Correct: 80% Incorrect: 20%
C (version 1)	18	Correct: 94% Incorrect: 6%	Correct: 89% Incorrect: 11%	Correct: 100% Incorrect: 0%	Correct: 100% Incorrect: 0%	Correct: 89% Incorrect: 11%
A (version 2)	8	Correct: 88% Incorrect: 12%	Correct: 75% Incorrect: 25%	Correct: 88% Incorrect: 12%	Correct: 75% Incorrect: 25%	Correct: 88% Incorrect: 12%
B* (version 2)	23	Correct: 87% Incorrect: 13%	Correct: 83% Incorrect: 17%	Correct: 78% Incorrect: 22%	Correct: 70% Incorrect: 30%	Correct: 83% Incorrect: 17%
C (version 2)	15	Correct: 100% Incorrect: 0%	Correct: 100% Incorrect: 0%	Correct: 87% Incorrect: 13%	Correct: 67% Incorrect: 33%	Correct: 80% Incorrect: 20%
Statistical Significance						
<p>Program State – Version 1: A one-way ANOVA indicated a significant difference ($p < 0.05$); But a Bernoulli test showed that these results are not significant.</p>						
<i>*Indicates two sections.</i>						

Table 31: Pennington’s Model: Environment Comparison (Version 1 vs. Version 2) – First Survey

Environment (version 1 and 2)	N	Elementary Operations	Control Flow	Data Flow	Program State	Program Function
IDLE (version 1)	15	Correct: 100% Incorrect: 0%	Correct: 85% Incorrect: 15%	Correct: 93% Incorrect: 7%	Correct: 80% Incorrect: 20%	Correct: 80% Incorrect: 20%
VIM (version 1)	44	Correct: 93% Incorrect: 7%	Correct: 95% Incorrect: 5%	Correct: 93% Incorrect: 7%	Correct: 86% Incorrect: 14%	Correct: 79% Incorrect: 21%
IDLE (version 2)	19	Correct: 84% Incorrect: 16%	Correct: 87% Incorrect: 13%	Correct: 83% Incorrect: 17%	Correct: 80% Incorrect: 20%	Correct: 88% Incorrect: 12%
VIM (version 2)	47	Correct: 98% Incorrect: 2%	Correct: 77% Incorrect: 23%	Correct: 91% Incorrect: 9%	Correct: 68% Incorrect: 32%	Correct: 83% Incorrect: 17%

Table 32: Pennington’s Model: Environment Comparison (Version 1 vs. Version 2) – Second Survey

Environment (version 1 and 2)	N	Elementary Operations	Control Flow	Data Flow	Program State	Program Function
IDLE (version 1)	19	Correct: 100% Incorrect: 0%	Correct: 100% Incorrect: 0%	Correct: 95% Incorrect: 5%	Correct: 95% Incorrect: 5%	Correct: 89% Incorrect: 11%
VIM (version 1)	22	Correct: 91% Incorrect: 9%	Correct: 86% Incorrect: 14%	Correct: 91% Incorrect: 9%	Correct: 95% Incorrect: 5%	Correct: 77% Incorrect: 23%
Both/Other (version 1)	11	Correct: 95% Incorrect: 5%	Correct: 82% Incorrect: 18%	Correct: 91% Incorrect: 9%	Correct: 73% Incorrect: 27%	Correct: 100% Incorrect: 0%
IDLE (version 2)	18	Correct: 89% Incorrect: 11%	Correct: 88% Incorrect: 12%	Correct: 83% Incorrect: 17%	Correct: 67% Incorrect: 33%	Correct: 88% Incorrect: 12%
VIM (version 2)	23	Correct: 100% Incorrect: 0%	Correct: 87% Incorrect: 13%	Correct: 78% Incorrect: 22%	Correct: 78% Incorrect: 22%	Correct: 78% Incorrect: 22%
Both/Other (version 2)	5	Correct: 60% Incorrect: 40%	Correct: 80% Incorrect: 20%	Correct: 100% Incorrect: 0%	Correct: 40% Incorrect: 60%	Correct: 80% Incorrect: 20%

Statistical Significance

Elementary Operations – Version 2: A one-way ANOVA indicated a significant different ($p < 0.01$); However none of the T-tests showed a significant difference. This may be due to the varying sample sizes.

Table 33: Pennington's Model: Version Comparison – First Survey

Version	N	Elementary Operations	Control Flow	Data Flow	Program State	Program Function
1	59	Correct: 95% Incorrect: 5%	Correct: 93% Incorrect: 7%	Correct: 93% Incorrect: 7%	Correct: 85% Incorrect: 15%	Correct: 79% Incorrect: 21%
2	63	Correct: 94% Incorrect: 6%	Correct: 79% Incorrect: 21%	Correct: 90% Incorrect: 10%	Correct: 71% Incorrect: 29%	Correct: 83% Incorrect: 17%
Statistical Significance						
Control Flow: A T-Test showed a significant difference ($p < 0.01$).						

Table 34: Pennington's Model: Version Comparison – Second Survey

Version	N	Elementary Operations	Control Flow	Data Flow	Program State	Program Function
1	52	Correct: 95% Incorrect: 5%	Correct: 90% Incorrect: 10%	Correct: 92% Incorrect: 8%	Correct: 90% Incorrect: 10%	Correct: 87% Incorrect: 13%
2	46	Correct: 91% Incorrect: 9%	Correct: 87% Incorrect: 13%	Correct: 83% Incorrect: 7%	Correct: 70% Incorrect: 30%	Correct: 83% Incorrect: 17%
Statistical Significance						
Program State: A T-Test showed a significant difference ($p = 0.01$).						

Table 35: Pennington’s Model: Changes in Understanding Programming Concepts – Section Comparison

Section (Version 1 and 2)	N	Elementary Operations	Control Flow	Data Flow	Program State	Program Function
A (version 1)	1st Survey - 14 2nd Survey - 9	1st Survey - 1.00 2nd Survey - 1.00	1st Survey - 0.86 2nd Survey - 0.89	1st Survey - 0.93 2nd Survey - 1.00	1st Survey - 0.79 2nd Survey - 0.67	1st Survey - 0.79 2nd Survey - 1.00
B* (version 1)	1st Survey - 27 2nd Survey - 25	1st Survey - 0.96 2nd Survey - 0.94	1st Survey - 0.93 2nd Survey - 0.92	1st Survey - 0.96 2nd Survey - 0.84	1st Survey - 0.81 2nd Survey - 0.92	1st Survey - 0.81 2nd Survey - 0.80
C (version 1)	1st Survey - 18 2nd Survey - 18	1st Survey - 0.89 2nd Survey - 0.94	1st Survey - 1.00 2nd Survey - 0.89	1st Survey - 0.89 2nd Survey - 1.00	1st Survey - 0.94 2nd Survey - 1.00	1st Survey - 0.76 2nd Survey - 0.89
A (version 2)	1st Survey - 12 2nd Survey - 8	1st Survey - 0.83 2nd Survey - 0.88	1st Survey - 0.83 2nd Survey - 0.75	1st Survey - 0.83 2nd Survey - 0.88	1st Survey - 0.92 2nd Survey - 0.75	1st Survey - 0.75 2nd Survey - 0.88
B* (version 2)	1st Survey - 31 2nd Survey - 23	1st Survey - 0.97 2nd Survey - 0.87	1st Survey - 0.68 2nd Survey - 0.83	1st Survey - 0.87 2nd Survey - 0.78	1st Survey - 0.65 2nd Survey - 0.70	1st Survey - 0.84 2nd Survey - 0.83
C (version 2)	1st Survey - 30 2nd Survey - 36	1st Survey - 0.95 2nd Survey - 1.00	1st Survey - 0.95 2nd Survey - 1.00	1st Survey - 1.00 2nd Survey - 0.87	1st Survey - 0.70 2nd Survey - 0.67	1st Survey - 0.85 2nd Survey - 0.80
<i>*Indicates two sections.</i>						

Table 36: Pennington's Model: Changes in Understanding Programming Concepts – Environment Comparison

Section (Version 1 and 2)	N	Elementary Operations	Control Flow	Data Flow	Program State	Program Function
IDLE (version 1)	1st Survey - 15	1st Survey - 1.00	1st Survey - 0.87	1st Survey - 0.93	1st Survey - 0.80	1st Survey - 0.80
	2nd Survey - 19	2nd Survey - 1.00	2nd Survey - 1.00	2nd Survey - 0.95	2nd Survey - 0.95	2nd Survey - 0.89
VIM (version 1)	1st Survey - 44	1st Survey - 0.93	1st Survey - 0.95	1st Survey - 0.93	1st Survey - 0.86	1st Survey - 0.79
	2nd Survey - 22	2nd Survey - 0.91	2nd Survey - 0.86	2nd Survey - 0.91	2nd Survey - 0.95	2nd Survey - 0.77
 						
IDLE (version 2)	1st Survey - 15	1st Survey - 0.87	1st Survey - 0.87	1st Survey - 0.93	1st Survey - 0.80	1st Survey - 0.87
	2nd Survey - 18	2nd Survey - 0.89	2nd Survey - 0.89	2nd Survey - 0.83	2nd Survey - 0.67	2nd Survey - 0.89
VIM (version 2)	1st Survey - 47	1st Survey - 0.98	1st Survey - 0.91	1st Survey - 0.87	1st Survey - 0.68	1st Survey - 0.83
	2nd Survey - 23	2nd Survey - 1.00	2nd Survey - 0.78	2nd Survey - 0.78	2nd Survey - 0.78	2nd Survey - 0.78

Table 37: Pennington's Model: Changes in Understanding Programming Concepts – Version Comparison

Version	N	Elementary Operations	Control Flow	Data Flow	Program State	Program Function
1	1st Survey - 59	1st Survey - 0.95	1st Survey - 0.93	1st Survey - 0.93	1st Survey - 0.85	1st Survey - 0.79
	2nd Survey - 52	2nd Survey - 0.95	2nd Survey - 0.90	2nd Survey - 0.92	2nd Survey - 0.90	2nd Survey - 0.87
2	1st Survey - 63	1st Survey - 0.94	1st Survey - 0.79	1st Survey - 0.90	1st Survey - 0.71	1st Survey - 0.83
	2nd Survey - 46	2nd Survey - 0.91	2nd Survey - 0.87	2nd Survey - 0.83	2nd Survey - 0.70	2nd Survey - 0.83

6.4.2 Protocol Analysis – “Think Aloud” Approach

A protocol analysis [43] was conducted to obtain both qualitative data and first-hand information about the students’ mental model for programming. The objective was to determine whether certain features within these environments can respectively shape the students’ mental model for programming. As previously mentioned, this study was conducted during the week of the environment switch. Therefore, the participants would be required to write a programming assignment using their new environment. The selection process for participants was based on random volunteers. There were seven students from either Section A or B who volunteered to participate in this study. Table 38 provides background information about each subject.

Similar to the assignment given during the CS1-Laboratory study, the subjects had to write a program that converted 700 days into y years, m months, and d days remaining. This task also required each subject to think aloud about their approach for writing this program. A video camera was used to record their feedback. Each subject was given 30 minutes to complete the assignment. The following subsections provide a summary of observations for each participant.

6.4.2.1 Summary of Observations – IDLE

Subject #1 – (S1)

After opening the IDLE environment, S1 began the assignment by creating a main method for his program. He then discussed logic and mathematical operations to make the appropriate conversions from the days given to the number of years, months, and days remaining. Next, he discussed how each conversion should have an output to the screen. Upon completing his code for the program, S1 was ready to check his solution. However, he was unsure about the procedures for interpreting a program in IDLE. He began to explore IDLE’s

Table 38: Background Information about Subjects [43]

Subject	Gender	Ethnicity	Programming Experience (prior to CS150)	Environment (after switch)
S1	M	C	None	IDLE
S2	M	C	HTML	VIM
S3	M	C	HTML	VIM
S4	F	AA	None	IDLE*
S5	F	C	None	IDLE
S6	F	AA	VIM**	VIM
S7	M	AA	VI, C++, Java, Fortran	VIM
<p><i>C = Caucasian; AA = African American;</i> <i>*Subject #4 was in an IDLE section (Section A) but chose to use VIM;</i> <i>**Subject 6 was repeating CS150;</i></p>				

menu options (on the menu bar) for possible solutions and would eventually find the *Run* option to interpret his program. S1 found four syntactical errors but managed to make necessary corrections to each. Overall, S1 would complete the assignment in eight minutes.

Subject #4 – (S4)

S4 needed immediate assistance with accessing IDLE. Afterwards, she discussed ideal logic and mathematical operations to make the appropriate conversions from the days given to the number of years, months, and days remaining. S4 immediately stated that she was lost. She expressed that using IDLE was not the concern, but rather approaching the programming assignment. S4 began writing code in the editing window, but continued to express that she was lost. She would ask for assistance from the facilitator. The facilitator handed her the alternate

copy of the instructions. At this point, S4 believed that she could complete the assignment. S4 began writing her code using the format of the example code. Upon finishing her code, she saved her program using the menu option from IDLE's menu bar. She was unsure about the procedure for interpreting her program, but would discover that option from IDLE's menu bar. S4 found several errors in her program and would struggle with correcting them. This would prevent S4 from completing the assignment.

Subject #5 – (S5)

After opening the IDLE environment, S5 began defining a function for converting days into years, remaining days into months, and days remaining. Afterwards, she wanted to save and interpret her current solution, but was not sure about the procedures for performing these actions respectively in IDLE. By using the Google search engine, S5 was able to find IDLE's website for assistance. However, she could only find information about saving a program as a python file. Therefore, she would use the Linux command terminal to locate and interpret her program file (like in VIM). S5 found multiple errors in her program, but managed to make the necessary corrections. Overall, S5 would complete the assignment in ~14 ½ minutes.

6.4.2.2 Summary of Observations – VIM

Subject #2 – (S2)

S2 needed immediate help accessing VIM through the Linux command terminal. He did not know any commands for VIM and began typing a snippet of code into the terminal. He expressed that IDLE usually allowed him to type and test a snippet of code, which was not the case in VIM. He then explored the terminal's menu bar for possible assistance with using VIM, but could not find any help. S2 started to type new code into the terminal while using its menu

options in hope to interpret his output, but was not successful. He mentioned that IDLE usually allowed him to interpret his program by selecting the *Run Module* option from its menu bar. S2 also attempted to use random Linux commands on the terminal, but failed to use them appropriately. He became frustrated and had to be assisted by the study's facilitator. S2 was handed two different sheets of VIM commands and an alternate version of the instructions that showed an example program for converting x minutes into h hours and m minutes remaining. S2 began typing random VIM commands from one of the sheets and eventually got one command to work. However, he did not understand the behavior of that command. After several attempts of using the VIM commands and not understanding their respective behavior, S2 became more frustrated and opened IDLE. He began to demonstrate how IDLE is easier for him to use. Therefore, S2 was never able to complete the assignment.

Subject #3 – (S3)

S3 needed immediate assistance with accessing VIM through the Linux command terminal. She immediately typed *help* in the command line for assistance, but only received a blank window in return. S3 attempted to type text in this window but nothing appeared. She reverted back to the VIM's menu bar for assistance on enabling text to appear in the terminal. After failing to find a solution, she closed the command terminal and opened a new terminal window. She pressed the *FI* key for the help menu, but still could not obtain any assistance. S3 would express a lack of confidence for understanding and using the VIM editor. She began typing code for the program in the terminal even though text did not appear. S3 would notice that the terminal has different modes (Visual/Insert), but did not understand their respective meanings. Therefore, she continued to write her code and would attempt to interpret her solution despite not being able to see the text. She checked the menu bar for the option to interpret her

program, but could not find this option. S3 believed that she may need two windows in order to run this program (like in IDLE). After being assisted by the facilitator and receiving two different sheets of VIM commands and the alternate version of the instructions, S3 continued to struggle with using the VIM editor and was never able to complete the assignment.

Subject #6 – (S6)

After opening the VIM editor, S6 began the assignment by creating a main method for her program. She then discussed logic and mathematical operations to make the appropriate conversions from the days given to the number of years, months, and days remaining. S6 admitted that she was lost and frustrated during this process. This would result in S6 continuously adding and removing code while attempting this assignment. S6 would receive assistance from the facilitator and was given the alternate copy of the instructions. After being assisted, S6 erased all of her code and began writing new code for her conversions. Upon finishing her code, S6 interpreted her program and received no syntactical errors. However, she noticed that her conversions were incorrect based on her output. She would struggle with determining the appropriate conversions for her program, which would prevent her from completing the assignment.

Subject #7 – (S7)

After opening the VIM editor, S7 used the *Help* command to open VIM's help menu to obtain further assistance with the environment. He stated that navigating through VIM was an issue since the mouse was not permitted. He also had trouble inserting text. At this point, he was tempted to use IDLE for this assignment. He mentioned that learning to use VIM was preventing him from attempting the assignment. However, he would figure out the procedure for inserting text into VIM. Afterwards, S7 discussed logic and mathematical operations to make the

appropriate conversions from the days given to the number of years, months, and days remaining. Before writing his code in VIM, he opened a Python console to write a snippet of code and test his solution. He interpreted his code and received an output. He then became concerned about how to perform these same actions using VIM. S7 would also open and test a previously written *HelloWorld* program in the console. He receives an output, but was still concerned about performing these actions in VIM. S7 was assisted by the facilitator and received two different sheets of VIM commands along with the alternate version of the instructions. S7 began typing the example code verbatim into the VIM editor. However, S7 would not complete the assignment.

6.4.2.3 Results

The subjects who originally learned VIM had less problems transitioning to IDLE. In particular, S1 and S5 were able to complete the assignment. Even though S4 and S6 used IDLE and VIM respectively during this study, their inability to complete the required task was due to the challenges of the assignment rather than these environments. On the other hand, the subjects who originally learned IDLE were not able to complete the assignment due to the challenges of using and understanding the VIM editor. Tables 39 and 40 summarize these results.

While working on the assignment, some of the subjects showed the tendency of reverting back to familiar procedures from their original environment if they felt lost or confused while using the new one. For example, S2 and S3 began using the menu bar of the command terminal assuming that VIM possessed features relative to IDLE. S5 began using the command terminal to interpret her program when she felt unsure about performing this procedure in IDLE.

Upon completing the study, each subject was asked about his or her preferred environment. Four of the six subjects, who used both environments, chose their original environment (Table 39). One reason was due to their acquired knowledge and experience with the original environment. Because of their prior experience, some subjects respectively felt that their original environment was quicker and easier to use.

Table 39: Task Completion Results [43]

Environment Transition	Subject	Completed Assignment	Preferred Environment
VIM to IDLE	S1	Yes	IDLE
	S4	No	VIM
	S5	Yes	VIM
IDLE to VIM	S2	No	IDLE
	S3	No	IDLE
	S7	No	VIM
VIM only	S6	No	VIM

Table 40: Challenges for NOT Completing Assignment [43]

Subject	Environment	Assignment
S2	X	
S3	X	
S4		X
S6		X
S7	X	

6.4.3 Program Procedures

The behavior of the IDLE users, who used VIM during the protocol analysis, raises a concern about the possible mental models novices acquire through using visual environments. As mentioned in Section 3.3, visual environments (like IDEs) can prevent novices from being exposed to the underlying factors of programming procedures. This type of mental model may also impose the challenge of learning other programming environments with contrasting feature sets, which was found to be true during the protocol analysis.

This section discusses a survey of open-ended questions that was given to the CS1 students to measure their understanding of programming procedures. This survey provided questions about writing, compiling, linking, and executing a program. The objective was to further explore these students' acquired mental models from using either IDLE or VIM. These questions were given twice during the semester (*once before the environment switch and once after the switch*).

The students' feedback from these questions was quantified in order to perform statistical analysis. Each answer was provided a weight based on the correctness of the students' response. Table 41 provides further detail about how each question was weighed. Three comparisons were used to discuss the results from this survey, which includes: *Section Comparison*, *Environment Comparison*, and *Question Comparison*. ANOVAs and T-Tests were used for each comparison. The following subsections detail the results of each comparison. (Refer back to *Section 2.5.3* for further details about Understanding Programming Procedures).

Table 41: Weights for Programming Procedures Survey

Standard Answers for each Question:

- **Explain process for creating a program** – *write code, compile code for errors, and execute for the output.*
- **Compilation** – *converts a program's source code into computer code.*
- **Linkage** – *creates an executable file from a successfully compiled file(s).*
- **Execution** – *instructions of a computer program are carried out; provides the output of a written program.*
- **Difference between Compiling/Executing and Interpreting a program** – *Interpretation shows the results or output of a program immediately without making an executable file. Compiling/Executing compiles a program, which eventually becomes an executable file and obtains its output.*

Weight of Score for each Question:

2 = entirely correct; 1 = partially correct; 0 = incorrect

6.4.3.1 Section Comparison

For the first survey, one-way ANOVAs and T-Tests were used to determine any significant differences between each section for providing an *entirely* correct answer for each question. The results indicated no significant difference for each question. Table 42 displays the responses for each question as *entirely correct*, *partially correct* or *incorrect/no response* and their respective percentages.

One-way ANOVAs and T-Tests were also used for the second survey to determine any significant differences between each section for providing an *entirely* correct answer for each question. One of the ANOVAs showed a significant difference for *explaining the process for*

creating a program ($p < 0.01$). This was possibly due to the lack of responses provided by the students in Section B, which caused a significant decrease in the overall average of responses for this particular section. T-tests also revealed a significant difference between sections A and B ($p < 0.01$) and sections B and C ($p < 0.01$) for possibly the same reason. Table 43 displays the responses for each question as *entirely correct*, *partially correct* or *incorrect/no response* and their respective percentages.

6.4.3.2 Environment Comparison

For the first survey, T-Tests were used to determine any significant differences between IDLE and VIM users (all sections) for providing an *entirely correct* answer for each question. One T-Test indicated a significant difference for *understanding compilation* ($p < 0.05$). Table 44 details the responses as *entirely correct*, *partially correct* or *incorrect/no response* and their respective percentages.

For the second survey, one-way ANOVAs and T-Tests were used to determine any significant differences between users of IDLE, VIM, or BOTH for providing an *entirely correct* answer for each question. The ANOVAs showed no significant differences for any of the questions. One T-Test indicated a significant difference for *explaining the procedure for creating a program* ($p < 0.05$). However, many of the students in Section B (who primarily used VIM) did not provide a response to this question. This may have caused a significant decrease in the overall average of responses for this particular section. Table 45 details the responses as *entirely correct*, *partially correct* or *incorrect/no response* and their respective percentages.

6.4.3.3 Question Comparison

A one-way ANOVA was used for both surveys to determine any significant differences amongst the five questions in regards to providing the correct answer. The results indicated a significant difference ($p < 0.01$). T-tests showed a significant difference between Explaining the Process of Creating a Program vs. Understanding Compilation ($p < 0.01$), Explaining the Process of Creating a Program vs. Understanding Linkage ($p < 0.01$), Explaining the Process of Creating a Program vs. Understanding the Difference Between Compiling/Executing and Interpreting a Program ($p < 0.01$), Understanding Compilation vs. Understanding Linkage ($p < 0.01$), Understanding Compilation vs. Understanding Execution ($p < 0.01$), Understanding Linkage vs. Understanding Execution ($p < 0.01$), Understanding Linkage vs. Understanding the Difference Between Compiling/Executing and Interpreting a Program ($p < 0.01$), and Understanding Execution vs. Understanding the Difference Between Compiling/Executing and Interpreting a Program ($p < 0.01$). Table 46 and 47 provides these results in further detail.

During the first survey, the majority provided an *entirely* correct response for explaining the process of creating a program along and understanding program execution. This was also true during the second survey assessment. The students however did not understand process of linking a program file. A common response to this question was that linking *puts multiple programs together*. Many of students also did not understand the difference between compiling/executing and interpreting a written program. This was also the case when measuring their understanding of program compilation.

6.4.3.4 First vs. Second Survey Comparison

When comparing the differences between sections for each question (Table 48), Section B showed a significant difference for *explaining the process of creating a program* ($p < 0.05$), *understanding compilation* ($p < 0.05$), and *understanding linkage* ($p < 0.01$). These significant differences may be influenced by the number of students who did not provide a response to these questions during the second assessment. These non-responses may have also influenced the significant differences between IDLE and VIM users as described in Table 49, where VIM users showed a significant decrease in the number of correct responses for *understanding compilation* ($p < 0.01$), *understanding linkage* ($p < 0.01$), and *understanding the difference between compiling/executing and interpreting a program* ($p < 0.01$). Section B's non-responses may have even influenced the results concerning the comparison between the correctness amongst each question (Table 50). The results showed a significant decrease after the second assessment for *understanding compilation* ($p = 0.01$), *understanding linkage* ($p < 0.01$), and *understanding the difference between compiling/executing and interpreting a program* ($p < 0.05$). Overall, the students showed consistency for understanding program execution. This was true when comparing sections, environments, and questions.

Table 42: Programming Procedures – Section Comparison (First Survey)

Section	A	B*	C
N	26	58	38
Explaining the Process of Creating a Program	Correct - Entirely: 65% Correct - Partially: 19% Incorrect/No Response: 15%	Correct - Entirely: 78% Correct - Partially: 16% Incorrect/No Response: 7%	Correct - Entirely: 79% Correct - Partially: 13% Incorrect/No Response: 8%
Understanding Compilation	Correct - Entirely: 35% Correct - Partially: 15% Incorrect/No Response: 50%	Correct - Entirely: 41% Correct - Partially: 19% Incorrect/No Response: 40%	Correct - Entirely: 47% Correct - Partially: 26% Incorrect/No Response: 26%
Understanding Linkage	Correct - Entirely: 8% Correct - Partially: 27% Incorrect/No Response: 65%	Correct - Entirely: 10% Correct - Partially: 43% Incorrect/No Response: 47%	Correct - Entirely: 3% Correct - Partially: 34% Incorrect/No Response: 63%
Understanding Execution	Correct - Entirely: 69% Correct - Partially: 8% Incorrect/No Response: 23%	Correct - Entirely: 71% Correct - Partially: 7% Incorrect/No Response: 22%	Correct - Entirely: 82% Correct - Partially: 3% Incorrect/No Response: 15%
Understanding the Difference Between Compiling/Executing and Interpreting a Program	Correct - Entirely: 54% Correct - Partially: 0% Incorrect/No Response: 46%	Correct - Entirely: 40% Correct - Partially: 9% Incorrect/No Response: 51%	Correct - Entirely: 34% Correct - Partially: 21% Incorrect/No Response: 45%
<i>*Indicates two sections.</i>			

Table 43: Programming Procedures – Section Comparison (Second Survey)

Section	A	B*	C
N	15	46	33
Explaining the Process of Creating a Program	Correct - Entirely: 87% Correct - Partially: 0% Incorrect/No Response: 13%	Correct - Entirely: 48% Correct - Partially: 50% Incorrect/No Response: 2%	Correct - Entirely: 85% Correct - Partially: 9% Incorrect/No Response: 6%
Understanding Compilation	Correct - Entirely: 20% Correct - Partially: 27% Incorrect/No Response: 53%	Correct - Entirely: 26% Correct - Partially: 7% Incorrect/No Response: 67%	Correct - Entirely: 45% Correct - Partially: 3% Incorrect/No Response: 52%
Understanding Linkage	Correct - Entirely: 13% Correct - Partially: 0% Incorrect/No Response: 87%	Correct - Entirely: 7% Correct - Partially: 0% Incorrect/No Response: 93%	Correct - Entirely: 9% Correct - Partially: 6% Incorrect/No Response: 85%
Understanding Execution	Correct - Entirely: 80% Correct - Partially: 0% Incorrect/No Response: 20%	Correct - Entirely: 78% Correct - Partially: 2% Incorrect/No Response: 20%	Correct - Entirely: 85% Correct - Partially: 3% Incorrect/No Response: 12%
Understanding the Difference Between Compiling/Executing and Interpreting a Program	Correct - Entirely: 13% Correct - Partially: 47% Incorrect/No Response: 40%	Correct - Entirely: 11% Correct - Partially: 37% Incorrect/No Response: 52%	Correct - Entirely: 18% Correct - Partially: 39% Incorrect/No Response: 42%
Statistical Significance			
Explaining the Process of Creating a Program: A one-way ANOVA indicated a significant difference ($p < 0.01$). T-tests showed a significant difference between Sections A and B ($p < 0.01$) and Sections C and B ($p < 0.01$).			
<i>*Indicates two sections.</i>			

Table 44: Programming Procedures – Environment Comparison (First Survey)

Environment	IDLE	VIM
N	30	92
Explaining the Process of Creating a Program	Correct - Entirely: 67% Correct - Partially: 17% Incorrect/No Response: 16%	Correct - Entirely: 78% Correct - Partially: 15% Incorrect/No Response: 7%
Understanding Compilation	Correct - Entirely: 27% Correct - Partially: 23% Incorrect/No Response: 50%	Correct - Entirely: 48% Correct - Partially: 20% Incorrect/No Response: 32%
Understanding Linkage	Correct - Entirely: 7% Correct - Partially: 27% Incorrect/No Response: 66%	Correct - Entirely: 8% Correct - Partially: 40% Incorrect/No Response: 52%
Understanding Execution	Correct - Entirely: 73% Correct - Partially: 7% Incorrect/No Response: 20%	Correct - Entirely: 74% Correct - Partially: 5% Incorrect/No Response: 21%
Understanding the Difference Between Compiling/Executing and Interpreting a Program	Correct - Entirely: 53% Correct - Partially: 0% Incorrect/No Response: 47%	Correct - Entirely: 37% Correct - Partially: 14% Incorrect/No Response: 49%
Statistical Significance		
Understanding Compilation: A T-test showed a significant difference between IDLE and VIM users ($p < 0.05$).		

Table 45: Programming Procedures – Environment Comparison (Second Survey)

Environment	IDLE	VIM	BOTH
N	36	45	13
Explaining the Process of Creating a Program	Correct - Entirely: 81% Correct - Partially: 19% Incorrect/No Response: 0%	Correct - Entirely: 58% Correct - Partially: 38% Incorrect/No Response: 4%	Correct - Entirely: 62% Correct - Partially: 31% Incorrect/No Response: 7%
Understanding Compilation	Correct - Entirely: 22% Correct - Partially: 19% Incorrect/No Response: 59%	Correct - Entirely: 36% Correct - Partially: 0% Incorrect/No Response: 64%	Correct - Entirely: 46% Correct - Partially: 8% Incorrect/No Response: 46%
Understanding Linkage	Correct - Entirely: 11% Correct - Partially: 0% Incorrect/No Response: 89%	Correct - Entirely: 9% Correct - Partially: 4% Incorrect/No Response: 87%	Correct - Entirely: 0% Correct - Partially: 0% Incorrect/No Response: 100%
Understanding Execution	Correct - Entirely: 83% Correct - Partially: 3% Incorrect/No Response: 14%	Correct - Entirely: 78% Correct - Partially: 0% Incorrect/No Response: 22%	Correct - Entirely: 85% Correct - Partially: 8% Incorrect/No Response: 8%
Understanding the Difference Between Compiling/Executing and Interpreting a Program	Correct - Entirely: 17% Correct - Partially: 44% Incorrect/No Response: 39%	Correct - Entirely: 9% Correct - Partially: 33% Incorrect/No Response: 58%	Correct - Entirely: 23% Correct - Partially: 46% Incorrect/No Response: 31%
Statistical Significance			
<p>Explaining the Process of Creating a Program: A T-test showed a significant difference between IDLE and VIM users ($p < 0.05$).</p>			

Table 46: Programming Procedures – Question Comparison (First Survey)

N	122
Explaining the Process of Creating a Program	Correct - Entirely: 75% Correct - Partially: 16% Incorrect/No Response: 9%
Understanding Compilation	Correct - Entirely: 42% Correct - Partially: 20% Incorrect/No Response: 38%
Understanding Linkage	Correct - Entirely: 7% Correct - Partially: 37% Incorrect/No Response: 57%
Understanding Execution	Correct - Entirely: 74% Correct - Partially: 6% Incorrect/No Response: 20%
Understanding the Difference Between Compiling/Executing and Interpreting a Program	Correct - Entirely: 41% Correct - Partially: 11% Incorrect/No Response: 48%

Statistical Significance

A one-way ANOVA indicated a significant difference ($p < 0.01$). T-tests indicated a significant difference for:

- Explaining the Process of Creating a Program vs. Understanding Compilation ($p < 0.01$).
- Explaining the Process of Creating a Program vs. Understanding Linkage ($p < 0.01$).
- Explaining the Process of Creating a Program vs. Understanding the Difference Between Compiling/Executing and Interpreting a Program ($p < 0.01$).
- Understanding Compilation vs. Understanding Linkage ($p < 0.01$).
- Understanding Compilation vs. Understanding Execution ($p < 0.01$).
- Understanding Linkage vs. Understanding Execution ($p < 0.01$).
- Understanding Linkage vs. Understanding the Difference Between Compiling/Executing and Interpreting a Program ($p < 0.01$).
- Understanding Execution vs. Understanding the Difference Between Compiling/Executing and Interpreting a Program ($p < 0.01$).

Table 47: Programming Procedures – Question Comparison (Second Survey)

N	94
Explaining the Process of Creating a Program	<p>Correct - Entirely: 67%</p> <p>Correct - Partially: 30%</p> <p>Incorrect/No Response: 3%</p>
Understanding Compilation	<p>Correct - Entirely: 32%</p> <p>Correct - Partially: 9%</p> <p>Incorrect/No Response: 59%</p>
Understanding Linkage	<p>Correct - Entirely: 9%</p> <p>Correct - Partially: 2%</p> <p>Incorrect/No Response: 89%</p>
Understanding Execution	<p>Correct - Entirely: 81%</p> <p>Correct - Partially: 2%</p> <p>Incorrect/No Response: 17%</p>
Understanding the Difference Between Compiling/Executing and Interpreting a Program	<p>Correct - Entirely: 14%</p> <p>Correct - Partially: 39%</p> <p>Incorrect/No Response: 47%</p>
<p style="text-align: center;">Statistical Significance</p> <p>A one-way ANOVA indicated a significant difference ($p < 0.01$). T-tests indicated a significant difference for:</p> <ul style="list-style-type: none"> • Explaining the Process of Creating a Program vs. Understanding Compilation ($p < 0.01$). • Explaining the Process of Creating a Program vs. Understanding Linkage ($p < 0.01$). • Explaining the Process of Creating a Program vs. Understanding the Difference Between Compiling/Executing and Interpreting a Program ($p < 0.01$). • Understanding Compilation vs. Understanding Linkage ($p < 0.01$). • Understanding Compilation vs. Understanding Execution ($p < 0.01$). • Understanding Linkage vs. Understanding Execution ($p < 0.01$). • Understanding Linkage vs. Understanding the Difference Between Compiling/Executing and Interpreting a Program ($p < 0.01$). • Understanding Execution vs. Understanding the Difference Between Compiling/Executing and Interpreting a Program ($p < 0.01$). 	

Table 48: Changes in Understanding Programming Procedures –Section Comparison

Section	A (averages: using weights discussed in Table 40)	B* (averages: using weights discussed in Table 40)	C (averages: using weights discussed in Table 40)
N	First Survey - 26 Second Survey - 15	First Survey - 58 Second Survey - 46	First Survey - 38 Second Survey - 33
Explaining the Process of Creating a Program	First Survey - 1.50 Second Survey - 1.87	First Survey - 1.71 Second Survey - 1.46	First Survey - 1.71 Second Survey - 1.79
Understanding Compilation	First Survey - 0.85 Second Survey - 0.67	First Survey - 1.02 Second Survey - 0.59	First Survey - 1.21 Second Survey - 0.94
Understanding Linkage	First Survey - 0.42 Second Survey - 0.27	First Survey - 0.64 Second Survey - 0.13	First Survey - 0.39 Second Survey - 0.24
Understanding Execution	First Survey - 1.46 Second Survey - 1.60	First Survey - 1.48 Second Survey - 1.59	First Survey - 1.66 Second Survey - 1.73
Understanding the Difference Between Compiling/Executing and Interpreting a Program	First Survey - 1.08 Second Survey - 0.73	First Survey - 0.88 Second Survey - 0.59	First Survey - 0.92 Second Survey - 0.76
Statistical Significance			
Explaining the Process of Creating a Program (Section B): T-Test showed a significant difference ($p < 0.05$).			
Understanding Compilation (Section B): T-Test showed a significant difference ($p < 0.05$).			
Understanding Linkage (Section B): T-Test showed a significant difference ($p < 0.01$).			
<i>*Indicates two sections.</i>			

Table 49: Changes in Understanding Programming Procedures –Environment Comparison

Questions	IDLE (averages: using weights discussed in Table 40)	VIM (averages: using weights discussed in Table 40)
N	First Survey - 30 Second Survey - 36	First Survey - 92 Second Survey - 45
Explaining the Process of Creating a Program	First Survey - 1.50 Second Survey - 1.81	First Survey - 1.72 Second Survey - 1.53
Understanding Compilation	First Survey - 0.77 Second Survey - 0.64	First Survey - 1.13 Second Survey - 0.71
Understanding Linkage	First Survey - 0.40 Second Survey - 0.22	First Survey - 0.55 Second Survey - 0.22
Understanding Execution	First Survey - 1.53 Second Survey - 1.69	First Survey - 1.53 Second Survey - 1.56
Understanding the Difference Between Compiling/Executing and Interpreting a Program	First Survey - 1.06 Second Survey - 0.78	First Survey - 0.89 Second Survey - 0.51
<p style="text-align: center;">Statistical Significance</p> <p>Understanding Compilation (VIM users): A T-test showed a significant difference ($p < 0.05$).</p> <p>Understanding Linkage (VIM users): A T-test showed a significant difference ($p < 0.01$).</p> <p>Understanding the Difference Between Compiling/Executing and Interpreting a Program (VIM users): A T-test showed a significant difference ($p < 0.01$).</p>		
<p style="text-align: center;"><i>*Excludes data of BOTH (VIM & IDLE) users.</i></p>		

Table 50: Changes in Understanding Programming Procedures – Question Comparison

Questions	Responses (averages: using weights discussed in Table 40)
Explaining the Process of Creating a Program	First Survey - 1.66 Second Survey - 1.64
Understanding Compilation	First Survey - 1.04 Second Survey - 0.72
Understanding Linkage	First Survey - 0.52 Second Survey - 0.19
Understanding Execution	First Survey - 1.53 Second Survey - 1.64
Understanding the Difference Between Compiling/Executing and Interpreting a Program	First Survey - 0.92 Second Survey - 0.67
<p style="text-align: center;">Statistical Significance</p> <p>Understanding Compilation: A T-test showed a significant difference ($p=0.01$).</p> <p>Understanding Linkage: A T-test showed a significant difference ($p<0.01$).</p> <p>Understanding the Difference Between Compiling/Executing and Interpreting a Program: A T-test showed a significant difference ($p<0.05$).</p>	

6.5 Time on Task

Time on task was measured during the four exams given throughout the semester (including the final exam). As part of each exam, the students were required to complete some programming tasks. Time on task for each exam was measured through a *proficiency rating*. This rating was used to control for students who completed their tasks quickly but managed to do poorly on the exam (and vice versa). To create a formula for calculating the proficiency rating, weights and constants were used. A student's *score on the exam* carried a heavier weight than *time on the exam*: **Proficiency Rating = ((14.6/(Time on Exam * 30))*(Score on Exam * 70))**.

Different approaches were used to control for the likelihood that some students did not consistently use their assigned environments throughout the semester. For Exam 0, a section comparison was used rather than an environment comparison while the Pennington's Model/Programming Procedure surveys along with email responses were used to determine the environments used on Exams 1, 2, and Final. However, there were students who did not participate in either the comprehension assessments or provide a response to the email sent in regards to their environment usage. These students were labeled UNKNOWN. The following subsections details each assessment.

6.5.1 Exam 0

The average proficiency rating for Exam 0 was 108 amongst all sections (Table 51). A one-way ANOVA and T-tests were used to determine any statistical significance. These tests showed no significant difference between the average proficiency ratings amongst the three

sections. Students in sections A and B showed a relatively close average proficiency rating of 106 and 107 respectively while Section C had a slightly higher rating of 112 (Figure 29).

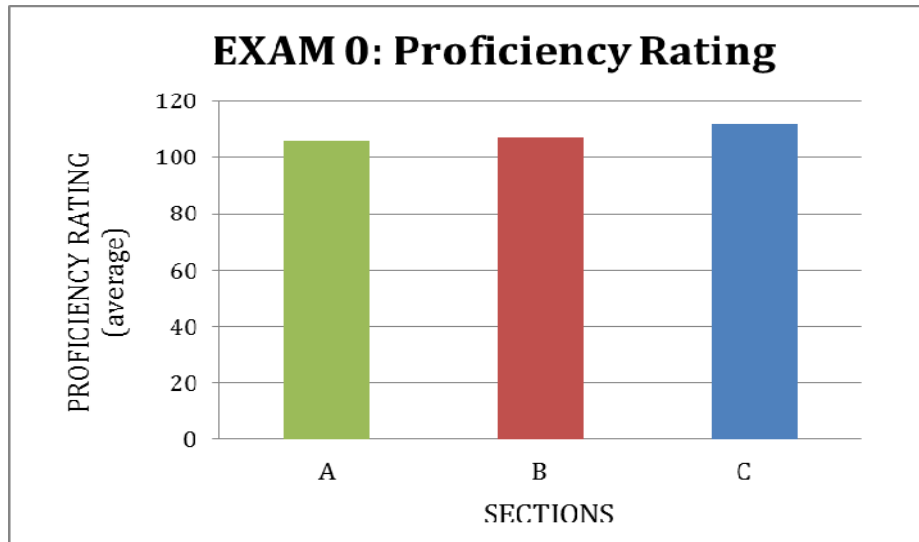


Figure 29: Exam 0 – Proficiency Rating (Section Comparison)

Table 51: Proficiency Rating Descriptive Data Amongst The Three Sections

Section	Average Rating	StdDev	Min Rating	Max Rating	N
A	106	30.85	15	126	41
B*	107	28.47	11	126	68
C	112	22.16	43	131	36
All Sections	108	27.70	11	131	145
<i>*Indicates two sections.</i>					

6.5.2 Exam 1

Section Comparison

The average proficiency rating for Exam 1 was 29 amongst all sections (Table 52). A one-way ANOVA and T-tests were used to determine any statistical significance. These tests showed no significant difference between the average proficiency ratings amongst the three sections; Section A had a score of 26, Section B had a score of 28, and Section C had a score of 35 (Figure 30).

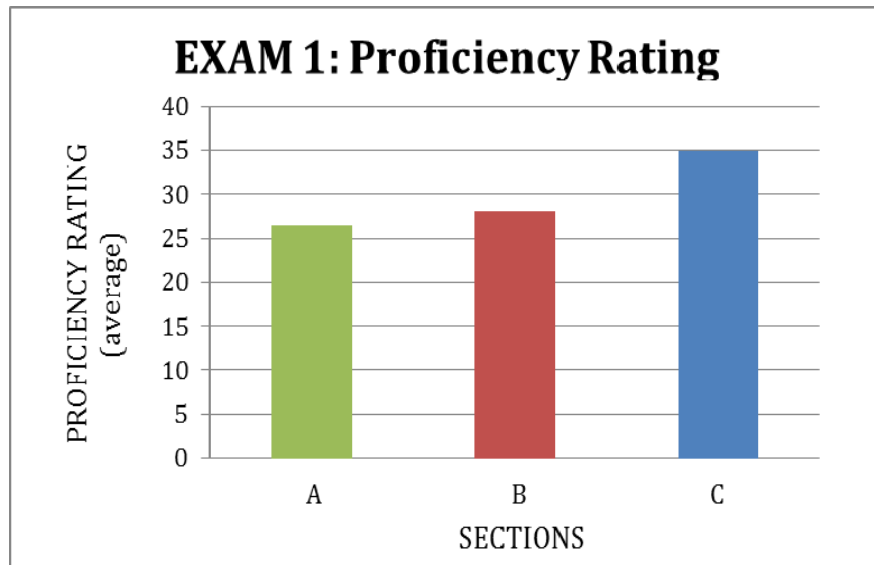


Figure 30: Exam 1 – Proficiency Rating (Section Comparison)

Table 52: Proficiency Rating Descriptive Data Amongst The Three Sections

Section	Average Rating	StdDev	Min Rating	Max Rating	N
A	26	11.22	6	52	36
B*	28	11.87	7	70	68
C	35	18.43	4	85	37
All Sections	29	14.06	4	85	141
<i>*Indicates two sections.</i>					

Environment Comparison

For Exam 1, the percentage of IDLE/VIM users varied amongst each section (Table 53). Majority of the students in Section A used IDLE (61%) while most students in sections B and C used VIM (78% for both). Overall, majority (61%) of the CS150 students used VIM on Exam 1.

Table 53: Percentage of IDLE/VIM Users – Exam 1		
Section	N	IDLE vs. VIM Users
A	36	IDLE - 61% VIM - 11% UNKNOWN - 28%
B*	68	IDLE - 0% VIM - 78% UNKNOWN - 22%
C	37	IDLE - 16% VIM - 78% UNKNOWN - 6%
All Sections	141	IDLE - 19% VIM - 61% UNKNOWN - 19%
<i>*Indicates two sections.</i>		

A proficiency rating was also applied based on the environments being used. A one-way ANOVA and T-tests were used to determine any statistical significance. These tests showed no significant difference between the average proficiency ratings amongst the environments. IDLE users had an average proficiency rating of 26, VIM users had a rating of 31, and UNKNOWN users scored a rating of 28 (Figure 31). Table 54 provides descriptive details of these results.

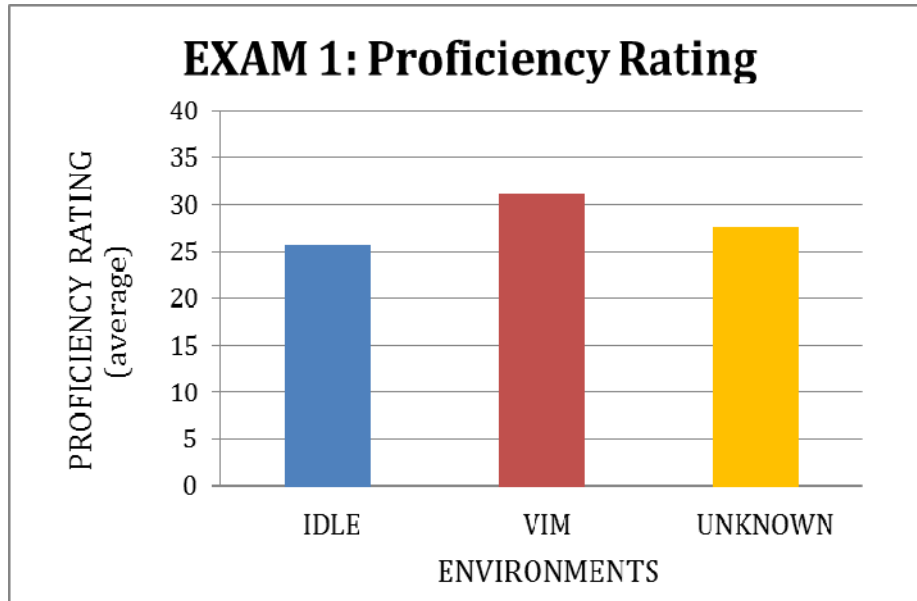


Figure 31: Exam 1 – Proficiency Rating (Environment Comparison)

Table 54: Proficiency Rating Descriptive Data Amongst the Environments

Environment	Average Rating	StdDev	Min Rating	Max Rating	N
IDLE	26	9.70	7	44	28
VIM	31	15.67	4	85	86
UNKNOWN	28	11.64	6	47	27
OVERALL	29	14.06	4	85	141

6.5.3 Exam 2

Section Comparison

The average proficiency rating for Exam 2 was 22 amongst all sections (Table 55). A one-way ANOVA and T-tests were used to determine any statistical significance. These tests showed no significant difference between the average proficiency ratings amongst the three

sections; Section A had a score of 20, Section B had a score of 21, and Section C had a score of 26 (Figure 32).

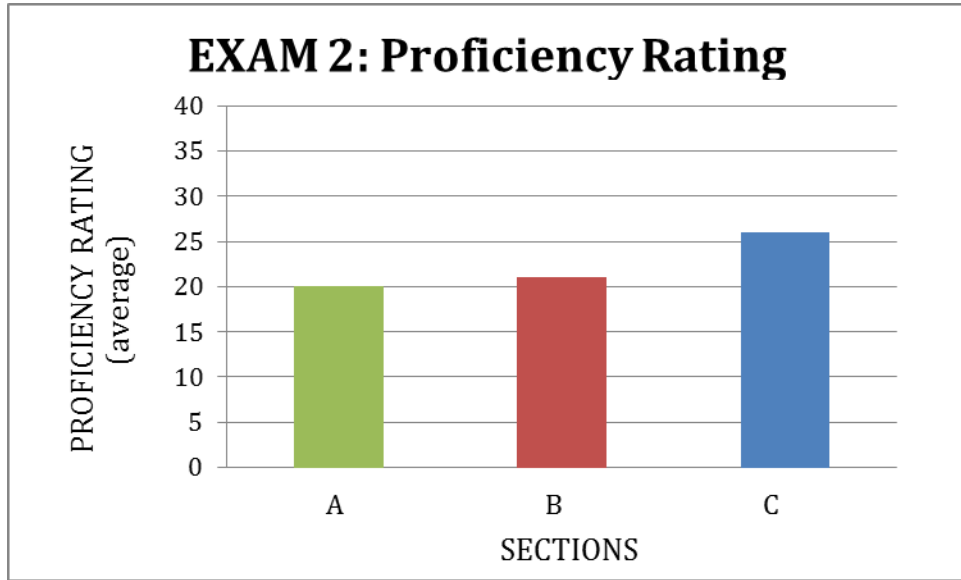


Figure 32: Exam 2 – Proficiency Rating (Section Comparison)

Table 55: Proficiency Rating Descriptive Data Amongst The Three Sections

Section	Average Rating	StdDev	Min Rating	Max Rating	N
A	20	15.65	2	67	33
B*	21	11.12	8	52	66
C	26	14.73	6	71	33
All Sections	22	13.41	2	71	132
<i>*Indicates two sections.</i>					

Environment Comparison

For Exam 2, the percentage of IDLE/VIM users varied amongst each section (Table 56). Majority of the students in Section A used IDLE (58%) while most students in sections B and C

used VIM (45% and 42% respectively). Overall, most (37%) of the CS150 students used VIM on Exam 2.

Table 56: Percentage of IDLE/VIM Users – Exam 2

Section	N	IDLE vs. VIM Users
A	33	IDLE - 58% VIM - 15% UNKNOWN - 27% BOTH - 0%
B*	66	IDLE - 18% VIM - 45% UNKNOWN - 27% BOTH - 9%
C	33	IDLE - 30% VIM - 42% UNKNOWN - 12% BOTH - 15%
All Sections	132	IDLE - 31% VIM - 37% UNKNOWN - 23% BOTH - 8%
<i>*Indicates two sections.</i>		

A one-way ANOVA and T-tests were used to determine any statistical significance between these environments for Exam 2. These tests showed no significant difference between the average proficiency ratings amongst the environments. IDLE users had an average proficiency rating of 24, VIM users had a rating of 21, UNKNOWN users scored a rating of 19, and users of BOTH (IDLE & VIM) showed a rating of 27 (Figure 33). Table 57 provides descriptive detail of these results.

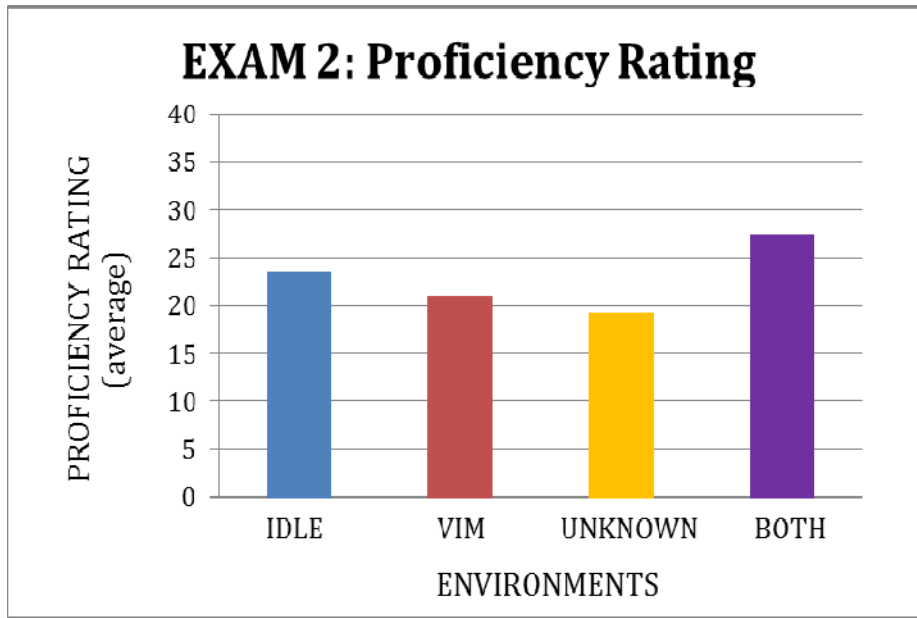


Figure 33: Exam 2 – Proficiency Rating (Environment Comparison)

Table 57: Proficiency Rating Descriptive Data Amongst the Environments

Environment	Average Rating	StdDev	Min Rating	Max Rating	N
IDLE	24	13.36	3	67	41
VIM	21	11.73	6	60	49
UNKNOWN	19	15.78	2	71	31
BOTH	27	12.83	6	46	11
OVERALL	22	13.41	2	71	132

6.5.4 Final Exam

Section Comparison

The average proficiency rating for the Final Exam was 19 amongst all sections (Table 58). A one-way ANOVA and T-tests were used to determine any statistical significance. The

ANOVA indicated a significant difference ($p < 0.01$). The T-tests showed significant difference between Sections A and C ($p < 0.01$) and Sections B and C ($p < 0.01$). Section A had an average proficiency rating of 16, Section B had a score of 17, and Section C showed a score of 27 (Figure 34).

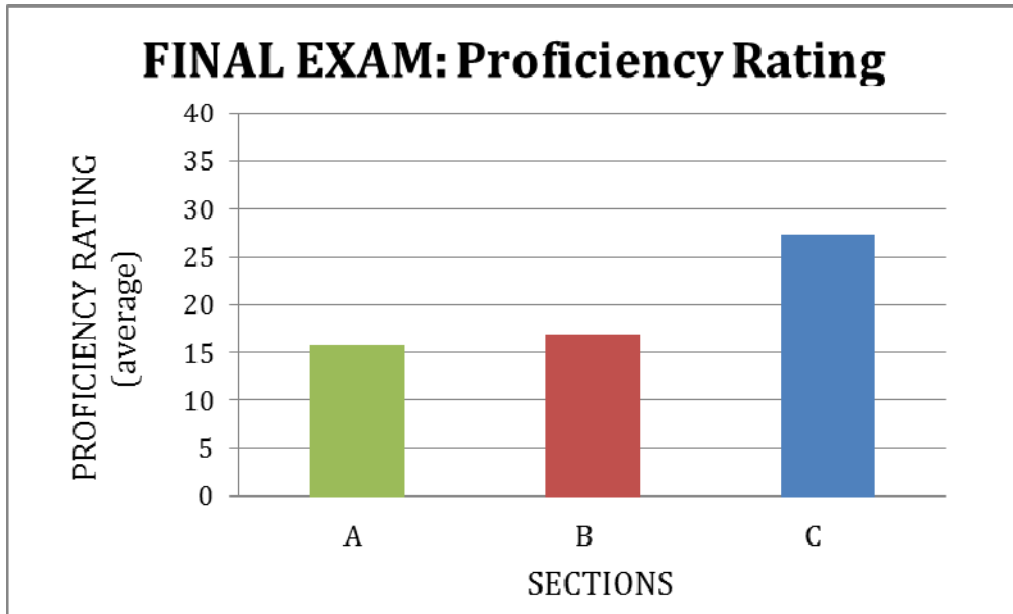


Figure 34: Final Exam – Proficiency Rating (Section Comparison)

Table 58: Proficiency Rating Descriptive Data Amongst The Three Sections

Section	Average Rating	StdDev	Min Rating	Max Rating	N
A	16	9.40	4	42	32
B*	17	10.80	1	47	63
C	27	16.34	4	85	35
All Sections	19	12.89	1	85	130
<i>*Indicates two sections.</i>					

Environment Comparison

For the Final Exam, the percentage of IDLE/VIM users varied amongst each section (Table 59). Majority of the students in Section A used IDLE (53%) while most students in sections B and C used VIM (44% and 46% respectively). Overall, most (38%) of the CS150 students used VIM on the Final Exam.

Table 59: Percentage of IDLE/VIM Users – Final Exam		
Section	N	IDLE vs. VIM Users
A	32	IDLE - 53% VIM - 16% UNKNOWN - 22% BOTH - 0%
B*	63	IDLE - 17% VIM - 44% UNKNOWN - 29% BOTH - 10%
C	35	IDLE - 31% VIM - 46% UNKNOWN - 11% BOTH - 11%
All Sections	130	IDLE - 32% VIM - 38% UNKNOWN - 22% BOTH - 8%
<i>*Indicates two sections.</i>		

A one-way ANOVA and T-tests were used to determine any statistical significance between these environments for the Final Exam. These tests showed no significant difference between the average proficiency ratings amongst the environments. IDLE users had an average proficiency rating of 21, VIM users had a rating of 17, UNKNOWN users scored a rating of 19,

and users of BOTH (IDLE & VIM) showed a rating of 26 (Figure 35). Table 60 provides descriptive detail of these results.

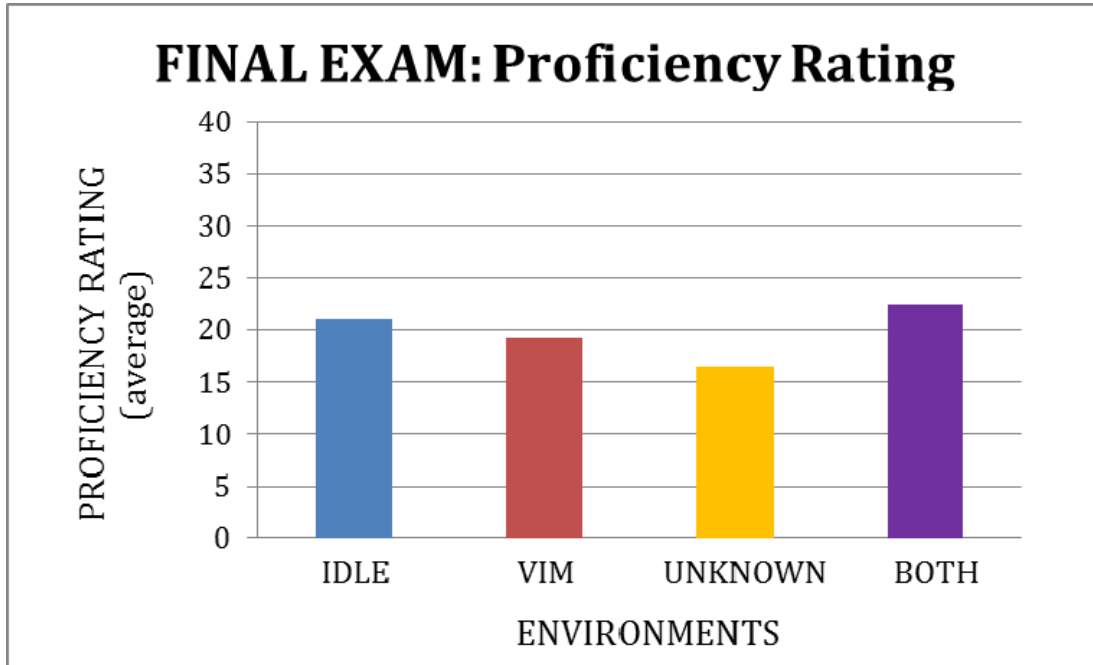


Figure 35: Final Exam – Proficiency Rating (Environment Comparison)

Table 60: Proficiency Rating Descriptive Data Amongst the Environments

Environment	Average Rating	StdDev	Min Rating	Max Rating	N
IDLE	21	10.97	5	56	42
VIM	17	14.32	1	85	49
UNKNOWN	19	13.75	2	64	29
BOTH	26	11.53	8	47	10
OVERALL	19	13.06	1	85	130

6.5.5 Trends for Proficiency Ratings

Throughout the duration of the semester, the average proficiency ratings for sections A and B respectively showed a continuous decrease. Section C slightly showed a steady average during the latter exams. There was a significant decrease in the average proficiency rating between Exam 0 and 1 for all sections ($p < 0.01$). A possible reason for this decrease is due to an increase in complexity of the material covered as the semester progressed. Exam 1 may have imposed a greater challenge to the students than Exam 0. Another significant decrease occurred between Exam 1 and 2 for all sections (*Sections A & C* – $p < 0.05$; *Section B* – $p < 0.01$). From Exam 2 to the Final Exam, only Section B showed a significant decrease in their average proficiency rating ($p < 0.05$). Figures 36a - c and illustrate this trend.

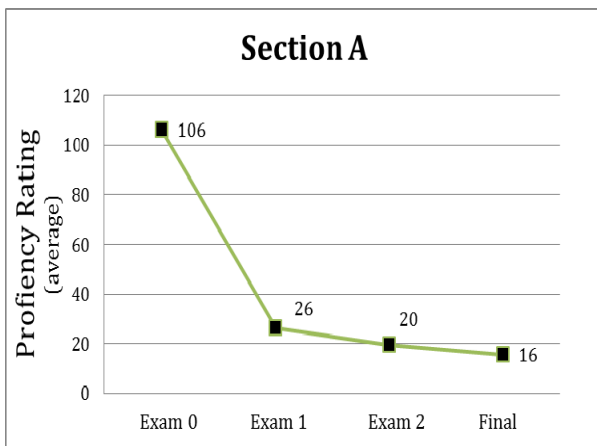


Figure36a. Proficiency ratings – Section A

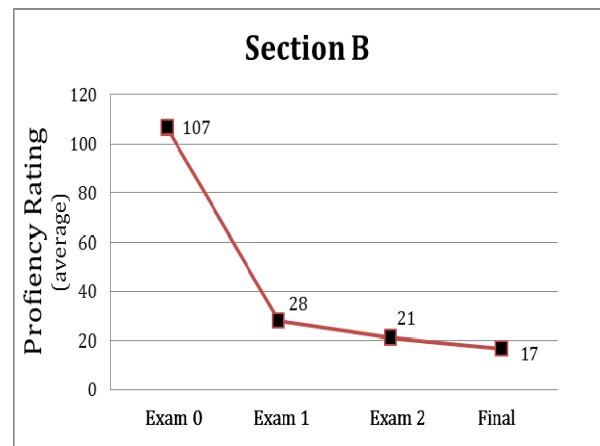


Figure36b. Proficiency ratings – Section B

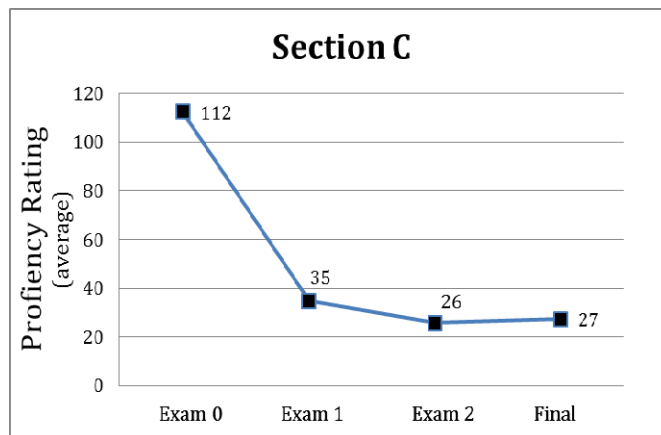


Figure36c. Proficiency ratings – Section C

When comparing environments, both IDLE and VIM showed a continuous decrease in their respective average proficiency ratings throughout the semester. The students, who were considered UNKNOWN, showed a steady rating during the latter exams. Students who used BOTH (IDLE & VIM) showed a slight decrease in their average rating from Exam 2 to the Final Exam.

IDLE users showed no significant decreases. VIM users only showed a significant decrease from Exam 1 to 2 ($p < 0.01$). This was also true for the UNKNOWN students ($p < 0.05$). Figures 37a - d and illustrate this trend.

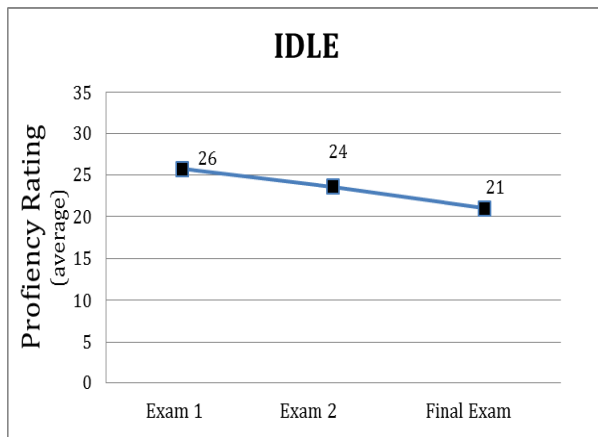


Figure37a. Proficiency ratings – IDLE

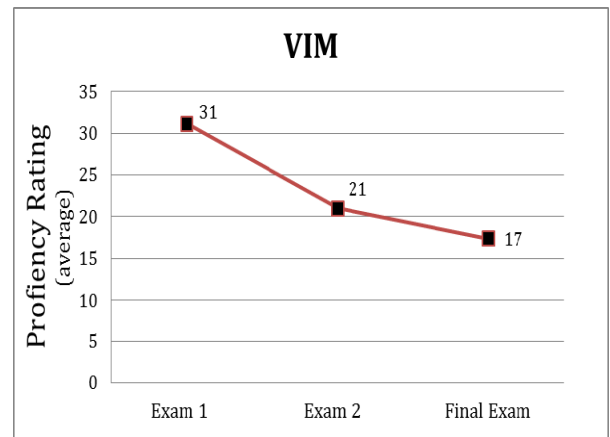


Figure37b. Proficiency ratings – VIM

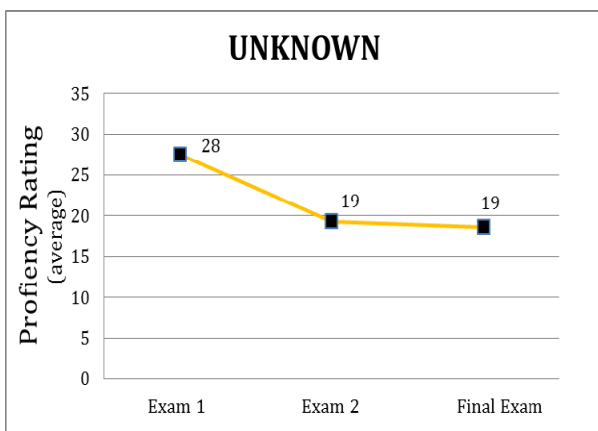


Figure37c. Proficiency ratings – UNKNOWN

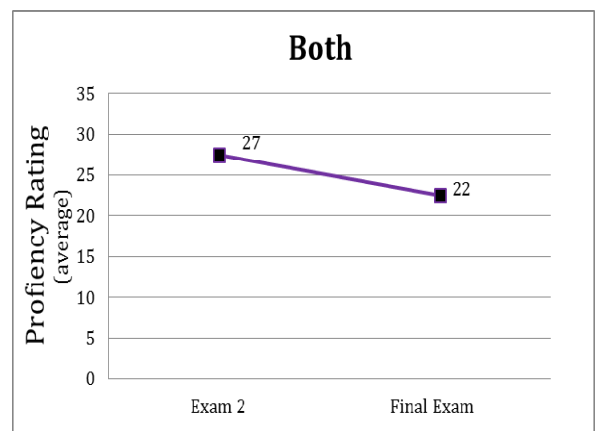


Figure37d. Proficiency ratings – BOTH

6.6 Usability Survey

A usability survey was issued to the students based on their assigned environment (IDLE or VIM). This survey allowed the students to provide feedback concerning their feelings about the assigned environments. Other questions required them to give feedback about their personal experience and the tools' attributes. This survey was administered three times during the semester (*twice before the environment switch and once after the switch*). Students in Section C were given the version of the survey that reflected the environment they were using. During the final assessment, students in Section C received both versions. To control for students who were using environments contrary to their assigned one, some questions provided the opportunity for them to indicate whether or not they were using the "assigned" environments.

The questions on this survey ranged from a students' initial impression with the assigned environment to a willingness of using it for projects independent of the course. The responses to these questions were either multiple choice or open-ended. The open-ended questions were quantified in order to conduct statistical analysis. One-way ANOVAs and T-tests were used as part of the analysis.

The assessments for this survey are categorized as First Survey, Second Survey, and Third Survey. The second and third surveys were altered to represent particular points in the semester. The following subsections detail the results from these assessments. A summary of these results is provided in Tables 61 - 67c.

6.6.1 First Survey

The student representation for the first survey was 119. Table 61 displays the students' responses as percentages for the entire group. Statistical significances and percentages for each section are displayed in Tables 62a - 62d. The results showed that students in sections A and B were less comfortable ($p < 0.01$) with IDLE and VIM respectively than students who used VIM in Section C. These sections also mishandled their respective environments more often than their peers, who used either IDLE ($p < 0.01$, including Bernoulli's test) or VIM (Section A: $p < 0.01$; Section B: $p < 0.05$), in Section C. In addition, sections A and B were found to be less confident with using their respective environments to complete another assignment (if necessary) than students, who used either IDLE ($p < 0.01$, including Bernoulli's test) or VIM ($p < 0.01$) in Section C. Students who used VIM in Section C liked this tool more than students in Section B ($p < 0.01$). They also liked VIM more than Section A students liked IDLE ($p < 0.01$). VIM students in Section C also showed a higher representation of students who would use their environment for random projects outside of a course than sections B ($p < 0.05$) and A - in respect to IDLE ($p < 0.01$).

Overall, the results from the first survey showed that students in Section C, in particular those using VIM, gave higher scores about their environments. One reason for their positive feedback may be due to expectations. By being an honor section, the expectations for these students were higher. Satisfying these expectations could have motivated them to obtain a better grasp of these environments than their peers in sections A and B.

Table 61: CS 150 Environment Usability Data – First Survey

Student Representation (N=119)			
Initial Impression	Comfort with “Assigned” Environment	Mishandling the “Assigned” Environment	Confident with Doing Another Assignment
Positive - 32% Non-Positive - 37% No Response - 31%	Not Comfortable At All - 3% Mostly Not Comfortable - 3% Slightly Comfortable - 6% 50/50 - 13% Fairly Comfortable - 24% Mostly Comfortable - 21% Absolutely Comfortable - 15% No Response - 16%	Absolutely Often - 3% Mostly Often - 8% Fairly Often - 13% 50/50 - 16% Slightly Often - 19% Mostly NOT Often - 23% Absolutely NOT Often - 2% No Response - 16%	Not Confident At All - 3% Mostly Not Confident - 3% Slightly Confident - 7% 50/50 - 14% Fairly Confident - 22% Mostly Confident - 16% Absolutely Confident - 19% No Response - 16%
Like the “Assigned” Environment	Easiest Attributes	Hardest Attributes	Prior Experience with other Environments (Besides IDLE or VIM)
Not At All - 3% Mostly Do Not Like - 4% Slightly Like - 6% 50/50 - 13% Fairly Like - 19% Mostly Like - 28% Absolutely Like - 12% No Response - 16%	Writing The Code/Python Attributes - 8% Environment Attributes - 55% Familiarity - 4% No Response/Nothing - 31% I Don’t Know - 1% Non-specific Response - 2%	Writing The Code/Python Attributes - 11% Environment Attributes - 41% Familiarity - 0% No Response/Nothing - 37% I Don’t Know - 2% Non-specific Response - 9%	Yes - 13% No - 29% No Response - 58%
Like the “Other” Environment (Including IDLE or VIM)	Prior Experience with Visual or Command Line Environments (Including IDLE & VIM)	“Assigned” Environment vs. “Other” Environment (Which do you like more?)	Use “Other” Environment for Random Projects Outside of a Course
Not At All - 1% Mostly Do Not Like - 1% Slightly Like - 2% 50/50 - 6% Fairly Like - 7% Mostly Like - 14% Absolutely Like - 8% No Response - 62%	Visual - 24% Command Line - 11% Non-specific Response - 3% No Response - 62%	Assigned Environment - 17% Other Environment - 18% Neither/Doesn’t Matter - 5% I Don’t Know - 0% No Response - 61%	Yes - 20% No - 15% I Don’t Know/Maybe - 0% No Response - 65%
Use “Assigned” Environment for Random Projects Outside of a Course			
Yes - 44% No - 19% I Don’t Know/Maybe - 3% No Response - 34%			

Table 62a: Section-by-Section Environment Usability Data – First Survey

Section	Initial Impression (IDLE)	Comfort with Environment (IDLE)	Mishandling the Environment (IDLE)
A (N=33)	Positive - 30% Non-Positive - 39% No Response - 30%	Not Comfortable At All - 6% Mostly Not Comfortable - 9% Slightly Comfortable - 6% 50/50 - 21% Fairly Comfortable - 24% Mostly Comfortable - 15% Absolutely Comfortable - 12% No Response - 6%	Absolutely Often - 3% Mostly Often - 15% Fairly Often - 9% 50/50 - 24% Slightly Often - 27% Mostly NOT Often - 15% Absolutely NOT Often - 6% No Response - 0%
C - IDLE (N=13)	Positive - 69% Non-Positive - 23% No Response - 8%	Not Comfortable At All - 0% Mostly Not Comfortable - 0% Slightly Comfortable - 8% 50/50 - 8% Fairly Comfortable - 15% Mostly Comfortable - 46% Absolutely Comfortable - 23% No Response - 0%	Absolutely Often - 0% Mostly Often - 0% Fairly Often - 8% 50/50 - 8% Slightly Often - 31% Mostly NOT Often - 46% Absolutely NOT Often - 8% No Response - 0%
Section	Initial Impression (VIM)	Comfort with Environment (VIM)	Mishandling the Environment (VIM)
B* (N=46)	Positive - 20% Non-Positive - 30% No Response - 50%	Not Comfortable At All - 0% Mostly Not Comfortable - 2% Slightly Comfortable - 7% 50/50 - 13% Fairly Comfortable - 26% Mostly Comfortable - 11% Absolutely Comfortable - 4% No Response - 37%	Absolutely Often - 2% Mostly Often - 9% Fairly Often - 13% 50/50 - 20% Slightly Often - 9% Mostly NOT Often - 11% Absolutely NOT Often - 0% No Response - 37%
C - VIM (N=27)	Positive – 37% Non-Positive – 52% No Response - 11%	Not Comfortable At All - 4% Mostly Not Comfortable - 0% Slightly Comfortable - 4% 50/50 - 4% Fairly Comfortable - 22% Mostly Comfortable - 33% Absolutely Comfortable - 33% No Response - 0%	Absolutely Often - 4% Mostly Often - 4% Fairly Often - 22% 50/50 - 4% Slightly Often - 22% Mostly NOT Often - 41% Absolutely NOT Often - 4% No Response - 0%
Statistical Significance			
<p>Comfort with Environment: A one-way ANOVA was conducted ($p < 0.01$); T-tests showed a significant difference between Sections C-VIM and B ($p < 0.01$) and Sections C-VIM and A ($p < 0.01$).</p> <p>Mishandling the Environment: A one-way ANOVA was conducted ($p < 0.01$); T-tests showed a significant difference between Sections C-IDLE and A ($p < 0.01$, including Bernoulli's test), Sections C-VIM and B ($p < 0.05$), and Sections C-IDLE and B ($p < 0.01$, including Bernoulli's test).</p>			
<i>*Indicates two sections.</i>			

Table 62b: Section-by-Section Environment Usability Data – First Survey (CONT'D)

Section	Confident with Doing Another Assignment (IDLE)	Like the Environment (IDLE)	Easiest Attributes (IDLE)
A (N=33)	Not Confident At All - 9% Mostly Not Confident - 3% Slightly Confident - 6% 50/50 - 24% Fairly Confident - 21% Mostly Confident - 15% Absolutely Confident - 15% No Response - 6%	Not At All - 9% Mostly Do Not Like - 9% Slightly Like - 18% 50/50 - 27% Fairly Like - 21% Mostly Like - 28% Absolutely Like - 9% No Response - 6%	Writing The Code/Python Attributes - 9% Environment Attributes - 54% Familiarity - 6% No Response/Nothing - 24% I Don't Know - 0% Non-specific Response - 6%
C - IDLE (N=13)	Not Confident At All - 0% Mostly Not Confident - 0% Slightly Confident - 8% 50/50 - 8% Fairly Confident - 15% Mostly Confident - 23% Absolutely Confident - 46% No Response - 0%	Not At All - 0% Mostly Do Not Like - 0% Slightly Like - 8% 50/50 - 15% Fairly Like - 31% Mostly Like - 31% Absolutely Like - 15% No Response - 0%	Writing The Code/Python Attributes - 0% Environment Attributes - 69% Familiarity - 15% No Response/Nothing - 15% I Don't Know - 0% Non-specific Response - 0%
Section	Confident with Doing Another Assignment(VIM)	Like the Environment (VIM)	Easiest Attributes (VIM)
B* (N=46)	Not Confident At All - 2% Mostly Not Confident - 2% Slightly Confident - 11% 50/50 - 13% Fairly Confident - 26% Mostly Confident - 2% Absolutely Confident - 7% No Response - 37%	Not At All - 0% Mostly Do Not Like - 4% Slightly Like - 11% 50/50 - 11% Fairly Like - 15% Mostly Like - 17% Absolutely Like - 4% No Response - 37%	Writing The Code/Python Attributes - 7% Environment Attributes - 43% Familiarity - 2% No Response/Nothing - 48% I Don't Know - 0% Non-specific Response - 0%
C - VIM (N=27)	Not Confident At All - 0% Mostly Not Confident - 4% Slightly Confident - 0% 50/50 - 7% Fairly Confident - 19% Mostly Confident - 37% Absolutely Confident - 33% No Response - 0%	Not At All - 0% Mostly Do Not Like - 0% Slightly Like - 4% 50/50 - 7% Fairly Like - 11% Mostly Like - 52% Absolutely Like - 26% No Response - 0%	Writing The Code/Python Attributes - 11% Environment Attributes - 67% Familiarity - 0% No Response/Nothing - 19% I Don't Know - 4% Non-specific Response - 0%
Statistical Significance			
Confident with Doing Another Assignment: A one-way ANOVA was conducted ($p < 0.01$); T-tests showed a significant difference between Sections C-IDLE and A ($p < 0.01$, including Bernoulli's test), Sections C-IDLE and B ($p < 0.01$, including Bernoulli's test), Sections C-VIM and A ($p < 0.01$) and Sections C-VIM and B ($p < 0.01$).			
Like the Environment: A one-way ANOVA was conducted ($p < 0.01$); T-tests showed a significant difference between Sections C-VIM and B ($p < 0.01$), and Sections C-VIM and A ($p < 0.01$).			
<i>*Indicates two sections.</i>			

Table 62c: Section-by-Section Environment Usability Data– First Survey (CONT'D)

Section	Hardest Attributes (IDLE)	Prior Experience with other Environments (Besides IDLE or VIM)	Like the “Other” Environment (Including VIM)
A (N=33)	Writing The Code/Python Attributes - 9% Environment Attributes - 39% Familiarity - 0% No Response/Nothing - 30% I Don’t Know - 0% Non-specific Response - 21%	Yes - 30% No - 18% No Response - 51%	Not At All - 0% Mostly Do Not Like - 0% Slightly Like - 0% 50/50 - 3% Fairly Like - 6% Mostly Like - 12% Absolutely Like - 12% No Response - 66%
C - IDLE (N=13)	Writing The Code/Python Attributes - 15% Environment Attributes - 31% Familiarity - 0% No Response/Nothing - 46% I Don’t Know - 8% Non-specific Response - 0%	Yes - 62% No - 23% No Response - 15%	Not At All - 8% Mostly Do Not Like - 0% Slightly Like - 0% 50/50 - 15% Fairly Like - 15% Mostly Like - 38% Absolutely Like - 8% No Response - 15%
Section	Hardest Attributes (VIM)	Prior Experience with other Environments (Besides IDLE or VIM)	Like the “Other” Environment (Including IDLE)
B* (N=46)	Writing The Code/Python Attributes - 11% Environment Attributes - 30% Familiarity - 0% No Response/Nothing - 50% I Don’t Know - 2% Non-specific Response - 7%	Yes - 7% No - 4% No Response - 89%	Not At All - 0% Mostly Do Not Like - 0% Slightly Like - 2% 50/50 - 0% Fairly Like - 0% Mostly Like - 4% Absolutely Like - 4% No Response - 89%
C - VIM (N=27)	Writing The Code/Python Attributes - 11% Environment Attributes - 67% Familiarity - 0% No Response/Nothing - 19% I Don’t Know - 0% Non-specific Response - 4%	Yes - 52% No - 15% No Response - 33%	Not At All - 0% Mostly Do Not Like - 4% Slightly Like - 4% 50/50 - 15% Fairly Like - 15% Mostly Like - 22% Absolutely Like - 7% No Response - 33%
<i>*Indicates two sections.</i>			

Table 62d: Section-by-Section Environment Usability Data – First Survey (CONT'D)

Section	Prior Experience with Visual or Command Line Environments (Including IDLE & VIM)	“Assigned” Environment vs. “Other” Environment (Which do you like more?)	Use “Other” Environment for Random Projects Outside of a Course	Use “Assigned” Environment for Random Projects Outside of a Course
A (N=33)	Visual - 15% Command Line - 15% Non-specific Response - 3% No Response - 66%	Assigned Environment - 9% Other Environment - 24% Neither/Doesn't Matter - 3% I Don't Know - 0% No Response - 63%	Yes - 21% No - 15% I Don't Know/Maybe - 0% No Response - 63%	Yes - 39% No - 36% I Don't Know/Maybe - 0% No Response - 24%
C - IDLE (N=13)	Visual - 8% Command Line - 62% Non-specific Response - 15% No Response - 15%	Assigned Environment - 38% Other Environment - 31% Neither/Doesn't Matter - 15% I Don't Know - 0% No Response - 15%	Yes - 46% No - 38% I Don't Know/Maybe - 0% No Response - 15%	Yes - 54% No - 31% I Don't Know/Maybe - 8% No Response - 8%
Section	Prior Experience with Visual or Command Line Environments (Including IDLE & VIM)	“Assigned” Environment vs. “Other” Environment (Which do you like more?)	Use “Other” Environment for Random Projects Outside of a Course	Use “Assigned” Environment for Random Projects Outside of a Course
B* (N=46)	Visual - 11% Command Line - 0% Non-specific Response - 0% No Response - 89%	Assigned Environment - 2% Other Environment - 7% Neither/Doesn't Matter - 2% I Don't Know - 0% No Response - 89%	Yes - 9% No - 2% I Don't Know/Maybe - 0% No Response - 89%	Yes - 24% No - 11% I Don't Know/Maybe - 4% No Response - 61%
C - VIM (N=27)	Visual - 63% Command Line - 0% Non-specific Response - 4% No Response - 33%	Assigned Environment - 41% Other Environment - 22% Neither/Doesn't Matter - 7% I Don't Know - 0% No Response - 30%	Yes - 26% No - 26% I Don't Know/Maybe - 0% No Response - 48%	Yes - 78% No - 7% I Don't Know/Maybe - 0% No Response - 15%
Statistical Significance				
Use “Assigned” Environment for Random Projects Outside of a Course: A one-way ANOVA was conducted ($p < 0.05$); T-tests showed a significant difference between Sections C-VIM and B ($p < 0.05$), and Sections C-VIM and A ($p < 0.01$).				
<i>*Indicates two sections.</i>				

6.6.2 Second Survey

The student representation for the second survey was also 119. Tables 63a and 63b display the students' responses as percentages for the entire group. These responses are also displayed by sections along with statistical significances in Tables 64a – 64f. The results showed that students in sections A and B were less comfortable with IDLE and VIM respectively than students who used either IDLE ($p < 0.01$, including *Bernoulli's test*) or VIM ($p < 0.01$) in Section C. These sections also mishandled their respective environments more often than their peers, who used VIM, in Section C (*Section A: $p < 0.05$; Section B: $p < 0.01$*). In addition, sections A and B were found to be with using their respective environments to complete another assignment (if necessary) than students, who used VIM, in Section C ($p < 0.01$). Students who used VIM in Section C liked this tool more than students in Section B ($p < 0.01$). They also liked VIM more than Section A liked IDLE ($p < 0.01$). When comparing their knowledge for using these environments (on a scale of 1-10), the VIM users in Section C scored higher than sections A ($p < 0.05$) and B ($p < 0.05$). Later questions regarding “other” environments showed no significant difference amongst the sections while some related questions received a limited amount of responses (see Tables 64c and 64d). Overall, the results from the second survey showed that students in Section C, in particular those using VIM, continued to give higher scores about the usability of their environments than sections A and B.

Table 63a: CS 150 Environment Usability Data – Second Survey

Student Representation (N=119)		
Comfort with “Assigned” Environment	Amount of Time to Become Comfortable with “Assigned” Environment	Mishandling the “Assigned” Environment
Not Comfortable At All - 2% Mostly Not Comfortable - 3% Slightly Comfortable - 3% 50/50 - 10% Fairly Comfortable - 14% Mostly Comfortable - 34% Absolutely Comfortable - 27% No Response - 6%	Still Not Comfortable - 8% 2 Months - 5% 1.5 Months - 4% 1 Month - 27% 2 to 3 Weeks - 36% 1 Week or Less- 10% Already Knew How to Use It - 4% No Response - 6%	Absolutely Often - 3% Mostly Often - 4% Fairly Often - 11% 50/50 - 20% Slightly Often - 34% Mostly NOT Often - 18% Absolutely NOT Often - 3% No Response - 6%
Confident with Doing Another Assignment	Like the “Assigned” Environment	Know How to Use “Assigned” Environment (on a scale of 1 – 10; N=103*)
Not Confident At All - 3% Mostly Not Confident - 3% Slightly Confident - 6% 50/50 - 19% Fairly Confident - 22% Mostly Confident - 20% Absolutely Confident - 21% No Response - 6%	Not At All - 6% Mostly Do Not Like - 3% Slightly Like - 5% 50/50 - 14% Fairly Like - 18% Mostly Like - 34% Absolutely Like - 13% No Response - 6%	Overall average: 6.99 <i>*Provided a response</i>
Prior Experience with other Environments (Besides IDLE or VIM)	Refer Back to the “Other” Environment to Complete Tasks while using the “Assigned” Environment (N= 48*)	Prior Experience with Visual or Command Line Environments (Including IDLE & VIM; N=51**)
Yes - 23% No - 18% No Response - 59%	Not At All - 27% Mostly No - 35% Slightly Yes - 13% 50/50 - 10% Fairly Yes - 6% Mostly Yes - 2% Absolutely Yes - 6% <i>*Provided a response</i>	Visual - 59% Command Line - 25% Non-specific Response - 16% <i>**Some students had prior experience with multiple environments</i>

Table 63b: CS 150 Environment Usability Data – Second Survey (CONT'D)

Student Representation (N=119)		
Comfort with “Other” Environment (N=49*)	Like the “Other” Environment (Including IDLE or VIM; N=49*)	Amount of Time to Become Comfortable with “Other” Environment (N=46*)
<p>Not Comfortable At All - 4% Mostly Not Comfortable - 6% Slightly Comfortable - 10% 50/50 - 12% Fairly Comfortable - 29% Mostly Comfortable - 22% Absolutely Comfortable - 16%</p> <p><i>*Provided a response</i></p>	<p>Not At All - 2% Mostly Do Not Like - 10% Slightly Like - 8% 50/50 - 22% Fairly Like - 20% Mostly Like - 18% Absolutely Like - 18%</p> <p><i>*Provided a response</i></p>	<p>Still Not Comfortable - 11% 2 Months - 7% 1.5 Months - 2% 1 Month - 28% 2 to 3 Weeks - 15% 1 Week or Less - 24% Already Knew How to Use It - 13%</p> <p><i>*Provided a response</i></p>
Mishandling the “Other” Environment (N=47*)	Know How to Use “Other” Environment (on a scale of 1 – 10; N=46*)	Refer Back to the “Assigned” Environment to Complete Tasks while using the “Other” Environment (N=49*)
<p>Absolutely Often - 6% Mostly Often - 2% Fairly Often - 17% 50/50 - 28% Slightly Often - 23% Mostly NOT Often - 23% Absolutely NOT Often - 0%</p> <p><i>*Provided a response</i></p>	<p>Overall average: 6.67</p> <p><i>*Provided a response</i></p>	<p>Not At All - 41% Mostly No - 16% Slightly Yes - 2% 50/50 - 16% Fairly Yes - 14% Mostly Yes - 6% Absolutely Yes - 4%</p> <p><i>*Provided a response</i></p>
“Assigned” Environment vs. “Other” Environment (Which do you like more?; N=45*)	Use “Other” Environment for Random Projects Outside of a Course (N=45*)	Use “Assigned” Environment for Random Projects Outside of a Course (N=90*)
<p>Assigned Environment - 67% Other Environment - 33% Neither/Doesn't Matter - 0% I Don't Know - 0%</p> <p><i>*Provided a response</i></p>	<p>Yes - 56% No - 40% I Don't Know/Maybe - 4%</p> <p><i>*Provided a response</i></p>	<p>Yes - 54% No - 33% I Don't Know/Maybe - 12%</p> <p><i>*Provided a response</i></p>

Table 64a: Section-by-Section Environment Usability Data – Second Survey

Section	Comfort with Environment (IDLE)	Amount of Time to Become Comfortable with Environment (IDLE)	Mishandling the Environment (IDLE)
A (N=28)	Not Comfortable At All - 4% Mostly Not Comfortable - 7% Slightly Comfortable - 0% 50/50 - 21% Fairly Comfortable - 14% Mostly Comfortable - 29% Absolutely Comfortable - 18% No Response - 7%	Still Not Comfortable - 14% 2 Months - 7% 1.5 Months - 4% 1 Month - 32% 2 to 3 Weeks - 14% 1 Week or Less- 11% Already Knew How to Use It - 11% No Response - 7%	Absolutely Often- 4% Mostly Often - 4% Fairly Often - 14% 50/50 - 21% Slightly Often - 32% Mostly NOT Often - 14% Absolutely NOT Often - 4% No Response - 7%
C - IDLE (N=7)	Not Comfortable At All - 0% Mostly Not Comfortable - 0% Slightly Comfortable - 0% 50/50 - 0% Fairly Comfortable - 0% Mostly Comfortable - 57% Absolutely Comfortable - 43% No Response - 0%	Still Not Comfortable - 0% 2 Months - 0% 1.5 Months - 0% 1 Month - 29% 2 to 3 Weeks - 57% 1 Week or Less- 14% Already Knew How to Use It - 0% No Response - 0%	Absolutely Often - 0% Mostly Often - 0% Fairly Often - 14% 50/50 - 14% Slightly Often - 0% Mostly NOT Often - 72% Absolutely NOT Often - 0% No Response - 0%
Section	Comfort with Environment (VIM)	Amount of Time to Become Comfortable with Environment (VIM)	Mishandling the Environment(VIM)
B* (N=53)	Not Comfortable At All - 2% Mostly Not Comfortable - 4% Slightly Comfortable - 8% 50/50 - 9% Fairly Comfortable - 23% Mostly Comfortable - 30% Absolutely Comfortable - 17% No Response - 8%	Still Not Comfortable - 9% 2 Months - 6% 1.5 Months - 4% 1 Month - 25% 2 to 3 Weeks - 38% 1 Week or Less- 11% Already Knew How to Use It - 0% No Response - 8%	Absolutely Often - 4% Mostly Often - 6% Fairly Often - 11% 50/50 - 25% Slightly Often - 40% Mostly NOT Often - 8% Absolutely NOT Often - 8% No Response - 0%
C - VIM (N=31)	Not Comfortable At All - 0% Mostly Not Comfortable - 0% Slightly Comfortable - 0% 50/50 - 3% Fairly Comfortable - 3% Mostly Comfortable - 42% Absolutely Comfortable - 48% No Response - 3%	Still Not Comfortable - 0% 2 Months - 3% 1.5 Months - 6% 1 Month - 26% 2 to 3 Weeks - 48% 1 Week or Less- 6% Already Knew How to Use It - 6% No Response - 3%	Absolutely Often - 0% Mostly Often - 3% Fairly Often - 6% 50/50 - 13% Slightly Often - 32% Mostly NOT Often - 32% Absolutely NOT Often - 10% No Response - 3%
Statistical Significance			
<p>Comfort with Environment: A one-way ANOVA was conducted ($p < 0.01$); T-tests showed a significant difference between Sections C-IDLE and A ($p < 0.01$, including <i>Bernoulli's test</i>), Sections C-IDLE and B ($p < 0.01$, including <i>Bernoulli's test</i>), Sections C-VIM and A ($p < 0.01$) and Sections C-VIM and B ($p < 0.01$).</p>			
<p>Mishandling the Environment: A one-way ANOVA was conducted ($p < 0.01$); T-tests showed a significant difference between Sections C-VIM and B ($p < 0.01$), and Sections C-VIM and A ($p < 0.05$).</p>			
<i>*Indicates two sections.</i>			

Table 64b: Section-by-Section Environment Usability Data – Second Survey (CONT'D)

Section	Confident with Doing Another Assignment (IDLE)	Like the Environment (IDLE)	Know How to Use Environment (IDLE)
A (N=28)	Not Confident At All - 7% Mostly Not Confident - 0% Slightly Confident - 7% 50/50 - 32% Fairly Confident - 25% Mostly Confident - 11% Absolutely Confident - 11% No Response - 7%	Not At All - 18% Mostly Do Not Like - 0% Slightly Like - 4% 50/50 - 21% Fairly Like - 11% Mostly Like - 29% Absolutely Like - 11% No Response - 7%	(N=23**) Overall average: 6.24 <i>**Provided a response</i>
C - IDLE (N=7)	Not Confident At All - 0% Mostly Not Confident - 14% Slightly Confident - 0% 50/50 - 14% Fairly Confident - 14% Mostly Confident - 43% Absolutely Confident - 14% No Response - 0%	Not At All - 0% Mostly Do Not Like - 0% Slightly Like - 0% 50/50 - 14% Fairly Like - 14% Mostly Like - 57% Absolutely Like - 14% No Response - 0%	(N=6**) Overall average: 7.33 <i>**Provided a response</i>
Section	Confident with Doing Another Assignment (VIM)	Like the Environment (VIM)	Know How to Use Environment (VIM)
B* (N=53)	Not Confident At All - 2% Mostly Not Confident - 6% Slightly Confident - 9% 50/50 - 17% Fairly Confident - 21% Mostly Confident - 26% Absolutely Confident - 11% No Response - 8%	Not At All - 4% Mostly Do Not Like - 8% Slightly Like - 6% 50/50 - 15% Fairly Like - 25% Mostly Like - 23% Absolutely Like - 13% No Response - 8%	(N=45**) Overall average: 6.92 <i>**Provided a response</i>
C - VIM (N=31)	Not Confident At All - 0% Mostly Not Confident - 0% Slightly Confident - 0% 50/50 - 13% Fairly Confident - 23% Mostly Confident - 13% Absolutely Confident - 48% No Response - 7%	Not At All - 0% Mostly Do Not Like - 0% Slightly Like - 6% 50/50 - 6% Fairly Like - 16% Mostly Like - 52% Absolutely Like - 16% No Response - 3%	(N=29**) Overall average: 7.62 <i>**Provided a response</i>
Statistical Significance			
<p>Confident with Doing Another Assignment: A one-way ANOVA was conducted ($p < 0.01$); T-tests showed a significant difference between Sections C-VIM and B ($p < 0.01$) and C-VIM and A ($p < 0.01$).</p> <p>Like the Environment: A one-way ANOVA was conducted ($p < 0.05$); T-tests showed a significant difference between Sections C-VIM and B ($p < 0.01$), and Sections C-VIM and A ($p < 0.01$).</p> <p>Know How to Use Environment: A one-way ANOVA was conducted ($p < 0.05$); T-tests showed a significant difference between Sections C-VIM and B ($p < 0.05$) and C-VIM and A ($p < 0.05$).</p>			
<i>*Indicates two sections.</i>			

Table 64c: Section-by-Section Environment Usability Data– Second Survey (CONT'D)

Section	Prior Experience with other Environments (Besides VIM)	Refer Back to the “Other” Environment to Complete Tasks while using IDLE	Prior Experience with Visual or Command Line Environments (Including VIM)
A (N=28)	Yes - 25% No - 14% No Response - 61%	(N=11**) Not At All - 36% Mostly No - 9% Slightly Yes - 0% 50/50 - 27% Fairly Yes - 9% Mostly Yes - 0% Absolutely Yes - 18% **Provided a response	(N=13***) Visual - 23% Command Line - 46% Non-specific Response - 31% ***Provided a response; Some students also had prior experience with multiple environments
C - IDLE (N=7)	Yes - 29% No - 43% No Response - 29%	(N=6**) Not At All - 33% Mostly No - 33% Slightly Yes - 17% 50/50 - 17% Fairly Yes - 0% Mostly Yes - 0% Absolutely Yes - 0% **Provided a response	(N=6**) Visual - 17% Command Line - 50% Non-specific Response - 33% **Provided a response
Section	Prior Experience with other Environments (Besides IDLE)	Refer Back to the “Other” Environment to Complete Tasks while using VIM	Prior Experience with Visual or Command Line Environments (Including IDLE)
B* (N=53)	Yes - 17% No - 4% No Response - 77%	(N=11**) Not At All - 0% Mostly No - 54% Slightly Yes - 27% 50/50 - 9% Fairly Yes - 0% Mostly Yes - 0% Absolutely Yes - 9% **Provided a response	(N=11**) Visual - 64% Command Line - 27% Non-specific Response - 9% **Provided a response
C - VIM (N=31)	Yes - 26% No - 42% No Response - 32%	(N=20**) Not At All - 35% Mostly No - 40% Slightly Yes - 10% 50/50 - 0% Fairly Yes - 10% Mostly Yes - 5% Absolutely Yes - 0% **Provided a response	(N=21**) Visual - 90% Command Line - 5% Non-specific Response - 5% **Provided a response
*Indicates two sections.			

Table 64d: Section-by-Section Environment Usability Data– Second Survey (CONT'D)

Section	Comfort with “Other” Environment	Like the “Other” Environment (Including VIM)	Amount of Time to Become Comfortable with “Other” Environment (Including VIM)
A (N=28)	(N=11**) Not Comfortable At All - 0% Mostly Not Comfortable - 9% Slightly Comfortable - 18% 50/50 - 27% Fairly Comfortable - 27% Mostly Comfortable - 0% Absolutely Comfortable - 18% <i>**Provided a response</i>	(N=11**) Not At All - 0% Mostly Do Not Like - 18% Slightly Like - 9% 50/50 - 36% Fairly Like - 9% Mostly Like - 0% Absolutely Like - 27% <i>**Provided a response</i>	(N=10**) Still Not Comfortable - 10% 2 Months - 0% 1.5 Months - 0% 1 Month - 40% 2 to 3 Weeks - 10% 1 Week or Less - 40% Already Knew How to Use It - 0% <i>**Provided a response</i>
C - IDLE (N=7)	(N=5**) Not Comfortable At All - 0% Mostly Not Comfortable - 20% Slightly Comfortable - 20% 50/50 - 0% Fairly Comfortable - 20% Mostly Comfortable - 0% Absolutely Comfortable - 40% <i>**Provided a response</i>	(N=5**) Not At All - 0% Mostly Do Not Like - 20% Slightly Like - 20% 50/50 - 0% Fairly Like - 20% Mostly Like - 20% Absolutely Like - 20% <i>**Provided a response</i>	(N=5**) Still Not Comfortable - 20% 2 Months - 20% 1.5 Months - 0% 1 Month - 20% 2 to 3 Weeks - 40% 1 Week or Less - 0% Already Knew How to Use It - 0% <i>**Provided a response</i>
Section	Comfort with “Other” Environment	Like the “Other” Environment (Including IDLE)	Amount of Time to Become Comfortable with “Other” Environment (Including IDLE)
B* (N=53)	(N=12**) Not Comfortable At All - 8% Mostly Not Comfortable - 0% Slightly Comfortable - 8% 50/50 - 8% Fairly Comfortable - 42% Mostly Comfortable - 8% Absolutely Comfortable - 25% <i>**Provided a response</i>	(N=12**) Not At All - 0% Mostly Do Not Like - 8% Slightly Like - 8% 50/50 - 8% Fairly Like - 33% Mostly Like - 25% Absolutely Like - 17% <i>**Provided a response</i>	(N=10**) Still Not Comfortable - 10% 2 Months - 0% 1.5 Months - 0% 1 Month - 20% 2 to 3 Weeks - 20% 1 Week or Less - 30% Already Knew How to Use It - 20% <i>**Provided a response</i>
C - VIM (N=31)	(N=21**) Not Comfortable At All - 5% Mostly Not Comfortable - 5% Slightly Comfortable - 5% 50/50 - 10% Fairly Comfortable - 24% Mostly Comfortable - 38% Absolutely Comfortable - 14% <i>**Provided a response</i>	(N=21**) Not At All - 5% Mostly Do Not Like - 5% Slightly Like - 5% 50/50 - 29% Fairly Like - 19% Mostly Like - 24% Absolutely Like - 14% <i>**Provided a response</i>	(N=21**) Still Not Comfortable - 10% 2 Months - 10% 1.5 Months - 5% 1 Month - 29% 2 to 3 Weeks - 10% 1 Week or Less - 19% Already Knew How to Use It - 19% <i>**Provided a response</i>

**Indicates two sections.*

Table 64e: Section-by-Section Environment Usability Data– Second Survey (CONT'D)

Section	Mishandling the “Other” Environment	Know How to Use “Other” Environment (on a scale of 1 – 10)	Refer Back to IDLE to Complete Tasks while using the “Other” Environment
A (N=28)	(N=11**) <ul style="list-style-type: none"> Absolutely Often - 9% Mostly Often - 0% Fairly Often - 18% 50/50 - 36% Slightly Often - 18% Mostly NOT Often - 18% Absolutely NOT Often - 0% **Provided a response	(N=10**) <ul style="list-style-type: none"> Overall average: 5.80 **Provided a response	(N=13**) <ul style="list-style-type: none"> Not At All - 46% Mostly No - 23% Slightly Yes - 8% 50/50 - 15% Fairly Yes - 8% Mostly Yes - 0% Absolutely Yes - 0% **Provided a response
C - IDLE (N=7)	(N=5**) <ul style="list-style-type: none"> Absolutely Often - 0% Mostly Often - 0% Fairly Often - 0% 50/50 - 0% Slightly Often - 80% Mostly NOT Often - 20% Absolutely NOT Often - 0% **Provided a response	(N=5**) <ul style="list-style-type: none"> Overall average: 5.60 **Provided a response	(N=6**) <ul style="list-style-type: none"> Not At All - 67% Mostly No - 0% Slightly Yes - 0% 50/50 - 17% Fairly Yes - 17% Mostly Yes - 0% Absolutely Yes - 0% **Provided a response
Section	Mishandling the “Other” Environment	Know How to Use “Other” Environment (on a scale of 1 – 10)	Refer Back to VIM to Complete Tasks while using the “Other” Environment
B* (N=53)	(N=11**) <ul style="list-style-type: none"> Absolutely Often - 0% Mostly Often - 0% Fairly Often - 18% 50/50 - 27% Slightly Often - 36% Mostly NOT Often - 18% Absolutely NOT Often - 0% **Provided a response	(N=11**) <ul style="list-style-type: none"> Overall average: 7.55 **Provided a response	(N=10**) <ul style="list-style-type: none"> Not At All - 40% Mostly No - 30% Slightly Yes - 0% 50/50 - 0% Fairly Yes - 20% Mostly Yes - 10% Absolutely Yes - 0% **Provided a response
C - VIM (N=31)	(N=20**) <ul style="list-style-type: none"> Absolutely Often - 10% Mostly Often - 5% Fairly Often - 20% 50/50 - 10% Slightly Often - 20% Mostly NOT Often - 35% Absolutely NOT Often - 0% **Provided a response	(N=20**) <ul style="list-style-type: none"> Overall average: 6.90 **Provided a response	(N=20**) <ul style="list-style-type: none"> Not At All - 30% Mostly No - 10% Slightly Yes - 0% 50/50 - 25% Fairly Yes - 15% Mostly Yes - 10% Absolutely Yes - 10% **Provided a response

*Indicates two sections.

Table 64f: Section-by-Section Environment Usability Data– Second Survey (CONT'D)

Section	IDLE vs. "Other" Environment (Which do you like more?)	Use "Other" Environment for Random Projects Outside of a Course	Use IDLE for Random Projects Outside of a Course
A (N=28)	(N=10**) Assigned Environment - 70% Other Environment - 30% Neither/Doesn't Matter - 0% I Don't Know - 0% **Provided a response	(N=10**) Yes - 30% No - 70% I Don't Know/Maybe - 0% **Provided a response	(N=23**) Yes - 39% No - 57% I Don't Know/Maybe - 4% **Provided a response
C - IDLE (N=7)	(N=5**) Assigned Environment - 80% Other Environment - 20% Neither/Doesn't Matter - 0% I Don't Know - 0% **Provided a response	(N=5**) Yes - 60% No - 40% I Don't Know/Maybe - 0% **Provided a response	(N=7**) Yes - 86% No - 14% I Don't Know/Maybe - 0% **Provided a response
Section	VIM vs. "Other" Environment (Which do you like more?)	Use "Other" Environment for Random Projects Outside of a Course	Use VIM for Random Projects Outside of a Course
B* (N=53)	(N=11**) Assigned Environment - 45% Other Environment - 54% Neither/Doesn't Matter - 0% I Don't Know - 0% **Provided a response	(N=10**) Yes - 70% No - 30% I Don't Know/Maybe - 0% **Provided a response	(N=35**) Yes - 51% No - 29% I Don't Know/Maybe - 20% **Provided a response
C - VIM (N=31)	(N=19**) Assigned Environment - 74% Other Environment - 26% Neither/Doesn't Matter - 0% I Don't Know - 0% **Provided a response	(N=20**) Yes - 60% No - 30% I Don't Know/Maybe - 10% **Provided a response	(N=25**) Yes - 64% No - 24% I Don't Know/Maybe - 12% **Provided a response

**Indicates two sections.*

6.6.3 Third Survey (*After Environment Switch*)

The student representation for the third survey was 122 (including a duplicate representation of Section C). Tables 65a and 65b display the students' responses as percentages for the entire group. These responses are also displayed by sections along with statistical significances in Tables 66a – 66f. The results showed that students in sections A and B were less comfortable with VIM and IDLE respectively than students in Section C, who used either IDLE (*Section A: $p < 0.01$, including Bernoulli's test; Section B: $p < 0.01$*) or VIM (*Section A: $p < 0.01$, including Bernoulli's test; Section B: $p < 0.01$*). Concerning the amount of time to become comfortable with these new environments, both sections also scored significantly lower than their peers in Section C. Section A scored lower than students in Section C, who used either IDLE (*$p < 0.01$, including Bernoulli's test*) or VIM (*$p < 0.01$, including Bernoulli's test*), while Section B scored lower than students in Section C, who used VIM (*$p < 0.01$*). These sections also mishandled their new environment more often than students in Section C, who used either IDLE (*Section A: $p < 0.01$, including Bernoulli's test; Section B: $p < 0.05$*) or VIM (*Section A: $p < 0.01$; Section B: $p < 0.05$*).

Section A showed the lowest confidence for using their new environment to complete another assignment (if necessary) than any other section (*$p < 0.01$, including Bernoulli's test*). Section B showed less confidence for the same question in comparison to students in Section C, who used either IDLE (*$p < 0.05$*) or VIM (*$p < 0.01$*). In addition, students using IDLE in Section C showed less confidence than their peers using VIM (*$p < 0.05$*) in the same section.

Section A liked using their new environment less than any other section (*$p < 0.01$, including Bernoulli's test*). Section B liked using their new environment less in comparison to students in Section C, who used VIM (*$p < 0.01$*). In addition, students who used IDLE in Section

C liked using this tool less than their peers who use VIM ($p < 0.05$) in the same section. When comparing their knowledge for using these environments (on a scale of 1-10), the IDLE and VIM users in Section C scored higher than sections A ($p < 0.01$, including *Bernoulli's test*) and B ($p < 0.01$) respectively.

For some questions regarding the original environments, it was assumed that many of the students' responses would reflect their "assigned" environment prior to the switch. Many of these questions indicated a significant difference between the sections. For example, sections A and B showed a greater possibility of referring back to their original environments while using the new one than students in Section C who used VIM (*Section A: $p < 0.01$, including Bernoulli's test; Section B: $p < 0.01$*). IDLE students in Section C also showed a greater possibility in comparison to those using VIM in the same section ($p < 0.05$). Sections A showed a higher comfort level for using their original environment than the new one in comparison to Section B ($p < 0.01$, including *Bernoulli's test*) and VIM students in Section C ($p < 0.01$, including *Bernoulli's test*). Section B showed a higher comfort level than students in Section C, who used either IDLE ($p < 0.01$) or VIM ($p < 0.01$). Sections A and B showed a higher fondness for their original environment than students in Section C, who used either IDLE (*Section A: $p < 0.01$, including Bernoulli's test; Section B: $p < 0.01$*) or VIM (*Section A: $p < 0.01$, including Bernoulli's test; Section B: $p < 0.05$*). In addition, Section A showed a higher fondness than Section B ($p < 0.01$, including *Bernoulli's test*). When comparing the new environment to the original environment, Section A showed a greater preference towards their original environment than VIM users in Section C ($p < 0.01$, including *Bernoulli's test*). Section B took a longer amount of time to become comfortable with the original environment than Section A ($p < 0.01$, including *Bernoulli's test*) and VIM students in Section C ($p < 0.01$). Section B also showed a greater tendency to mishandle their original

environment than Section A ($p < 0.01$, including Bernoulli's test) and IDLE students in Section C ($p < 0.05$). Section A showed a greater knowledge of using their original environment than Section B ($p < 0.01$, including Bernoulli's test) and IDLE users in Section C ($p < 0.01$, including Bernoulli's test). Section B also showed a greater knowledge than both IDLE ($p < 0.01$) and VIM ($p < 0.05$) users respectively in Section C. Section A showed a lower possibility of referring back to the new "assigned" environment while using the original one than any other section ($p < 0.01$, including Bernoulli's test). Sections A and B showed a greater possibility of using their original environment for random projects outside of class than both IDLE ($p < 0.01$, including Bernoulli's test) and VIM ($p < 0.01$, including Bernoulli's test) students in Section C. On the other hand, VIM users in Section C showed a greater possibility of using this tool than the "assigned" environment amongst all sections (Section A: $p < 0.01$, including Bernoulli's test; Section B & C-IDLE: $p < 0.05$).

Overall, students in sections A and B showed some differences in their experience with using their new environments. In many cases, both sections respectively gave lower scores than students in Section C concerning their new environments. There were also cases where Section A scored significantly lower than Section B on similar questions. However, these sections gave higher scores on questions about their original environment than Section C. These responses indicate that Sections A and B possibly showed a preference to their original environment, which in many cases was the originally "assigned" environments prior to the switch. The next section provides more detail about the difference in sections A and B's usability score before and after the switch.

Table 65a: CS 150 Environment Usability Data – Third Survey (After Environment Switch)

Student Representation (N=122)		
Comfort with New Environment	Amount of Time to Become Comfortable with New Environment	Mishandling the New Environment
Not Comfortable At All - 5% Mostly Not Comfortable - 7% Slightly Comfortable - 12% 50/50 - 16% Fairly Comfortable - 20% Mostly Comfortable - 15% Absolutely Comfortable - 21% No Response - 4%	Still Not Comfortable - 20% 2 Months - 4% 1.5 Months - 2% 1 Month - 25% 2 to 3 Weeks - 22% 1 Week or Less- 17% Already Knew How to Use It - 5% No Response - 4%	Absolutely Often - 5% Mostly Often - 6% Fairly Often - 8% 50/50 - 20% Slightly Often - 19% Mostly NOT Often - 31% Absolutely NOT Often - 6% No Response - 6%
Confident with Doing Another Assignment	Like the New Environment	Know How to Use New Environment (on a scale of 1 – 10; N=103*)
Not Confident At All - 7% Mostly Not Confident - 6% Slightly Confident - 7% 50/50 - 17% Fairly Confident - 20% Mostly Confident - 14% Absolutely Confident - 23% No Response - 5%	Not At All - 11% Mostly Do Not Like - 11% Slightly Like - 7% 50/50 - 20% Fairly Like - 18% Mostly Like - 14% Absolutely Like - 13% No Response - 5%	Overall average: 5.96 <i>*Provided a response</i>
Prior Experience with other Environments (Besides IDLE or VIM)	Refer Back to the “Other” Environment to Complete Tasks while using the New Environment (Including IDLE or VIM)	Prior Experience with Visual or Command Line Environments (Besides IDLE or VIM; N=16*)
Yes - 13% No - 87%	Not At All - 19% Mostly No - 14% Slightly Yes - 5% 50/50 - 18% Fairly Yes - 19% Mostly Yes - 5% Absolutely Yes - 11% No Response - 10%	Visual - 81% Command Line - 19% Non-specific Response - 0% <i>*Provided a response</i>

Table 65b: CS 150 Environment Usability Data – Third Survey (After Environment Switch) (CONT'D)

Student Representation (N=122)		
Comfort with “Other” Environment (Including IDLE or VIM)	Like the “Other” Environment (Including IDLE or VIM)	Amount of Time to Become Comfortable with “Other” Environment (Including IDLE or VIM)
Not Comfortable At All - 1% Mostly Not Comfortable - 2% Slightly Comfortable - 7% 50/50 - 14% Fairly Comfortable - 18% Mostly Comfortable - 24% Absolutely Comfortable - 21% No Response - 12%	Not At All - 2% Mostly Do Not Like - 5% Slightly Like - 4% 50/50 - 20% Fairly Like - 20% Mostly Like - 25% Absolutely Like - 11% No Response - 11%	Still Not Comfortable - 3% 2 Months - 9% 1.5 Months - 7% 1 Month - 30% 2 to 3 Weeks - 22% 1 Week or Less - 14% Already Knew How to Use It - 3% No Response - 11%
Mishandling the “Other” Environment (Including IDLE or VIM)	Know How to Use “Other” Environment (Including IDLE or VIM; on a scale of 1 – 10; N=91*)	Refer Back to the New Environment to Complete Tasks while using the “Other” Environment (N=106*)
Absolutely Often - 3% Mostly Often - 3% Fairly Often - 11% 50/50 - 18% Slightly Often - 20% Mostly NOT Often - 29% Absolutely NOT Often - 3% No Response - 12%	Overall average: 7.08 <i>*Provided a response</i>	Not At All - 34% Mostly No - 23% Slightly Yes - 4% 50/50 - 21% Fairly Yes - 9% Mostly Yes - 7% Absolutely Yes - 3% <i>*Provided a response</i>
New Environment vs. “Other” Environment (Which do you like more?; N=94*)	Use “Other” Environment for Random Projects Outside of a Course (N=76*)	Use New Environment for Random Projects Outside of a Course (N=99*)
New Environment - 34% Other Environment - 57% Neither/Doesn't Matter - 9% I Don't Know - 0% <i>*Provided a response</i>	Yes - 78% No - 17% I Don't Know/Maybe - 5% <i>*Provided a response</i>	Yes - 43% No - 48% I Don't Know/Maybe - 8% <i>*Provided a response</i>

Table 66a: Section-by-Section Environment Usability Data – Third Survey (After Environment Switch)

Section	Comfort with Environment (VIM)	Amount of Time to Become Comfortable with Environment (VIM)	Mishandling the Environment (VIM)
A (N=13)	Not Comfortable At All - 15% Mostly Not Comfortable - 15% Slightly Comfortable - 23% 50/50 - 38% Fairly Comfortable - 8% Mostly Comfortable - 0% Absolutely Comfortable - 0% No Response - 0%	Still Not Comfortable - 69% 2 Months - 0% 1.5 Months - 0% 1 Month - 15% 2 to 3 Weeks - 0% 1 Week or Less- 15% Already Knew How to Use It - 0% No Response - 0%	Absolutely Often- 23% Mostly Often - 23% Fairly Often - 15% 50/50 - 8% Slightly Often - 23% Mostly NOT Often - 8% Absolutely NOT Often - 0% No Response - 0%
C - IDLE (N=32)	Not Comfortable At All - 3% Mostly Not Comfortable - 0% Slightly Comfortable - 6% 50/50 - 18% Fairly Comfortable - 27% Mostly Comfortable - 12% Absolutely Comfortable - 30% No Response - 3%	Still Not Comfortable - 9% 2 Months - 0% 1.5 Months - 6% 1 Month - 33% 2 to 3 Weeks - 24% 1 Week or Less- 18% Already Knew How to Use It - 6% No Response - 3%	Absolutely Often- 3% Mostly Often - 0% Fairly Often - 6% 50/50 - 12% Slightly Often - 30% Mostly NOT Often - 36% Absolutely NOT Often - 9% No Response - 3%
Section	Comfort with Environment (IDLE)	Amount of Time to Become Comfortable with Environment (IDLE)	Mishandling the Environment (IDLE)
B* (N=41)	Not Comfortable At All - 7% Mostly Not Comfortable - 11% Slightly Comfortable - 20% 50/50 - 13% Fairly Comfortable - 20% Mostly Comfortable - 11% Absolutely Comfortable - 9% No Response - 9%	Still Not Comfortable - 27% 2 Months - 2% 1.5 Months - 2% 1 Month - 20% 2 to 3 Weeks - 16% 1 Week or Less- 20% Already Knew How to Use It - 4% No Response - 9%	Absolutely Often- 4% Mostly Often - 9% Fairly Often - 4% 50/50 - 29% Slightly Often - 16% Mostly NOT Often - 18% Absolutely NOT Often - 7% No Response - 13%
C - VIM (N=31)	Not Comfortable At All - 0% Mostly Not Comfortable - 3% Slightly Comfortable - 3% 50/50 - 6% Fairly Comfortable - 19% Mostly Comfortable - 29% Absolutely Comfortable - 33% No Response - 0%	Still Not Comfortable - 0% 2 Months - 13% 1.5 Months - 0% 1 Month - 29% 2 to 3 Weeks - 39% 1 Week or Less- 13% Already Knew How to Use It - 6% No Response - 0%	Absolutely Often- 0% Mostly Often - 0% Fairly Often - 13% 50/50 - 19% Slightly Often - 10% Mostly NOT Often - 55% Absolutely NOT Often - 3% No Response - 0%
Statistical Significance			
<p>Comfort with Environment: A one-way ANOVA was conducted ($p < 0.01$); T-tests showed a significant difference between Sections C-IDLE and A ($p < 0.01$, including <i>Bernoulli's test</i>), Sections C-IDLE and B ($p < 0.01$), Sections C-VIM and A ($p < 0.01$, including <i>Bernoulli's test</i>) and Sections C-VIM and B ($p < 0.01$).</p>			
<p>Amount of Time to Become Comfortable with Environment: A one-way ANOVA was conducted ($p < 0.01$); T-tests showed a significant difference between Sections C-IDLE and A ($p < 0.01$, including <i>Bernoulli's test</i>), Sections C-VIM and A ($p < 0.01$, including <i>Bernoulli's test</i>) and Sections C-VIM and B ($p < 0.01$).</p>			
<p>Mishandling the Environment: A one-way ANOVA was conducted ($p < 0.01$); T-tests showed a significant difference between Sections C-IDLE and A ($p < 0.01$, including <i>Bernoulli's test</i>), C-VIM and B ($p < 0.05$), C-IDLE and B ($p < 0.05$) and Sections C-VIM and A ($p < 0.01$).</p>			
<i>*Indicates two sections.</i>			

Table 66b: Section-by-Section Environment Usability Data – Third Survey (After Environment Switch) (CONT'D)

Section	Confident with Doing Another Assignment (IDLE)	Like the Environment (IDLE)	Know How to Use Environment (IDLE)
A (N=13)	Not Confident At All - 31% Mostly Not Confident - 23% Slightly Confident - 8% 50/50 - 31% Fairly Confident - 0% Mostly Confident - 8% Absolutely Confident - 0% No Response - 0%	Not At All - 46% Mostly Do Not Like - 15% Slightly Like - 8% 50/50 - 23% Fairly Like - 8% Mostly Like - 0% Absolutely Like - 0% No Response - 0%	(N=12*) Overall average: 3.92 *Provided a response
C - IDLE (N=32)	Not Confident At All - 3% Mostly Not Confident - 3% Slightly Confident - 9% 50/50 - 15% Fairly Confident - 18% Mostly Confident - 21% Absolutely Confident - 27% No Response - 3%	Not At All - 9% Mostly Do Not Like - 21% Slightly Like - 9% 50/50 - 12% Fairly Like - 18% Mostly Like - 15% Absolutely Like - 12% No Response - 3%	(N=26*) Overall average: 6.90 *Provided a response
Section	Confident with Doing Another Assignment (VIM)	Like the Environment (VIM)	Know How to Use Environment (VIM)
B* (N=41)	Not Confident At All - 9% Mostly Not Confident - 4% Slightly Confident - 11% 50/50 - 22% Fairly Confident - 27% Mostly Confident - 4% Absolutely Confident - 11% No Response - 11%	Not At All - 11% Mostly Do Not Like - 7% Slightly Like - 7% 50/50 - 27% Fairly Like - 18% Mostly Like - 11% Absolutely Like - 9% No Response - 11%	(N=39*) Overall average: 5.20 *Provided a response
C - VIM (N=31)	Not Confident At All - 0% Mostly Not Confident - 3% Slightly Confident - 0% 50/50 - 6% Fairly Confident - 23% Mostly Confident - 23% Absolutely Confident - 45% No Response - 0%	Not At All - 0% Mostly Do Not Like - 6% Slightly Like - 6% 50/50 - 16% Fairly Like - 23% Mostly Like - 23% Absolutely Like - 26% No Response - 0%	(N=26*) Overall average: 7.21 *Provided a response
Statistical Significance			
<p>Confident with Doing Another Assignment: A one-way ANOVA was conducted ($p < 0.01$); T-tests showed a significant difference between Sections C-IDLE and A ($p < 0.01$, including Bernoulli's test), Sections C-VIM and B ($p < 0.01$), Sections C-VIM and C-IDLE ($p < 0.05$), Sections C-IDLE and B ($p < 0.05$), C-VIM and A ($p < 0.01$, including Bernoulli's test) and Sections B and A ($p < 0.01$, including Bernoulli's test).</p>			
<p>Like the Environment: A one-way ANOVA was conducted ($p < 0.01$); T-tests showed a significant difference between Sections C-IDLE and A ($p < 0.01$, including Bernoulli's test), Sections C-VIM and B ($p < 0.01$), Sections C-VIM and C-IDLE ($p < 0.01$), C-VIM and A ($p < 0.01$, including Bernoulli's test) and Sections B and A ($p < 0.01$, including Bernoulli's test).</p>			
<p>Know How to Use Environment: A one-way ANOVA was conducted ($p < 0.01$); T-tests showed a significant difference between Sections C-IDLE and A ($p < 0.01$, including Bernoulli's test), Sections C-IDLE and B ($p < 0.01$), Sections C-VIM and B ($p < 0.01$), and C-VIM and A ($p < 0.01$, including Bernoulli's test).</p>			
*Indicates two sections.			

Table 66c: Section-by-Section Environment Usability Data – Third Survey (After Environment Switch) (CONT'D)

Section	Prior Experience with other Environments (Besides VIM)	Refer Back to the “Other” Environment to Complete Tasks while using the New Environment (Including IDLE)	Prior Experience with Visual or Command Line Environments (Besides IDLE)
A (N=13)	Yes - 8% No - 92% No Response - 0%	Not At All - 0% Mostly No - 15% Slightly Yes - 8% 50/50 - 8% Fairly Yes - 23% Mostly Yes - 0% Absolutely Yes - 46% No Response - 0%	(N=1**) Visual - 100% Command Line - 0% Non-specific Response - 0% **Provided a response
C - IDLE (N=32)	Yes - 15% No - 85% No Response - 0%	Not At All - 21% Mostly No - 9% Slightly Yes - 6% 50/50 - 24% Fairly Yes - 21% Mostly Yes - 9% Absolutely Yes - 3% No Response - 6%	(N=5**) Visual - 100% Command Line - 0% Non-specific Response - 0% **Provided a response
Section	Prior Experience with other Environments (Besides IDLE or VIM)	Refer Back to the “Other” Environment to Complete Tasks while using the New Environment (Including VIM)	Prior Experience with Visual or Command Line Environments (Besides VIM)
B* (N=41)	Yes - 11% No - 89% No Response - 0%	Not At All - 13% Mostly No - 13% Slightly Yes - 2% 50/50 - 16% Fairly Yes - 22% Mostly Yes - 4% Absolutely Yes - 13% No Response - 16%	(N=5**) Visual - 40% Command Line - 60% Non-specific Response - 0% **Provided a response
C - VIM (N=31)	Yes - 16% No - 84% No Response - 0%	Not At All - 32% Mostly No - 19% Slightly Yes - 6% 50/50 - 19% Fairly Yes - 10% Mostly Yes - 3% Absolutely Yes - 0% No Response - 10%	(N=5**) Visual - 100% Command Line - 0% Non-specific Response - 0% **Provided a response
Statistical Significance			
Refer Back to the “Other” Environment to Complete Tasks while using the New Environment: A one-way ANOVA was conducted ($p < 0.01$); T-tests showed a significant difference between Sections C-VIM and A ($p < 0.01$, including Bernoulli's test) and Sections C-IDLE and C-VIM ($p < 0.05$).			
*Indicates two sections.			

Table 66d: Section-by-Section Environment Usability Data – Third Survey (After Environment Switch) (CONT'D)

Section	Comfort with “Other” Environment (Including IDLE)	Like the “Other” Environment (Including IDLE)	Amount of Time to Become Comfortable with “Other” Environment (Including IDLE)
A (N=13)	Not Comfortable At All - 0% Mostly Not Comfortable - 0% Slightly Comfortable - 0% 50/50 - 0% Fairly Comfortable - 0% Mostly Comfortable - 54% Absolutely Comfortable - 38% No Response - 8%	Not At All - 0% Mostly Do Not Like - 0% Slightly Like - 0% 50/50 - 0% Fairly Like - 0% Mostly Like - 69% Absolutely Like - 23% No Response - 8%	Still Not Comfortable - 0% 2 Months - 0% 1.5 Months - 15% 1 Month - 0% 2 to 3 Weeks - 38% 1 Week or Less- 38% Already Knew How to Use It - 0% No Response - 8%
C - IDLE (N=32)	Not Comfortable At All - 0% Mostly Not Comfortable - 3% Slightly Comfortable - 6% 50/50 - 6% Fairly Comfortable - 21% Mostly Comfortable - 27% Absolutely Comfortable - 30% No Response - 6%	Not At All - 0% Mostly Do Not Like - 6% Slightly Like - 0% 50/50 - 21% Fairly Like - 27% Mostly Like - 27% Absolutely Like - 15% No Response - 3%	Still Not Comfortable - 0% 2 Months - 15% 1.5 Months - 3% 1 Month - 33% 2 to 3 Weeks - 33% 1 Week or Less- 9% Already Knew How to Use It - 3% No Response - 3%
Section	Comfort with “Other” Environment (Including VIM)	Like the “Other” Environment (Including VIM)	Amount of Time to Become Comfortable with “Other” Environment (Including VIM)
B* (N=41)	Not Comfortable At All - 2% Mostly Not Comfortable - 4% Slightly Comfortable - 9% 50/50 - 24% Fairly Comfortable - 18% Mostly Comfortable - 16% Absolutely Comfortable - 4% No Response - 22%	Not At All - 7% Mostly Do Not Like - 4% Slightly Like - 7% 50/50 - 27% Fairly Like - 20% Mostly Like - 9% Absolutely Like - 2% No Response - 23%	Still Not Comfortable - 9% 2 Months - 11% 1.5 Months - 7% 1 Month - 33% 2 to 3 Weeks - 9% 1 Week or Less- 9% Already Knew How to Use It - 2% No Response - 20%
C - VIM (N=31)	Not Comfortable At All - 0% Mostly Not Comfortable - 0% Slightly Comfortable - 10% 50/50 - 13% Fairly Comfortable - 23% Mostly Comfortable - 19% Absolutely Comfortable - 29% No Response - 6%	Not At All - 0% Mostly Do Not Like - 7% Slightly Like - 7% 50/50 - 19% Fairly Like - 23% Mostly Like - 23% Absolutely Like - 13% No Response - 10%	Still Not Comfortable - 0% 2 Months - 3% 1.5 Months - 10% 1 Month - 32% 2 to 3 Weeks - 23% 1 Week or Less- 16% Already Knew How to Use It - 6% No Response - 10%
Statistical Significance			
<p>Comfort with “Other” Environment: A one-way ANOVA was conducted ($p < 0.01$); T-tests showed a significant difference between Sections C-VIM and A ($p < 0.01$, including Bernoulli’s test), Sections B and A ($p < 0.01$, including Bernoulli’s test), Sections C-IDLE and B ($p < 0.01$), and Sections C-VIM and B ($p < 0.01$).</p>			
<p>Like the “Other” Environment: A one-way ANOVA was conducted ($p < 0.01$); T-tests showed a significant difference between Sections C-IDLE and A ($p < 0.01$, including Bernoulli’s test), Sections C-VIM and B ($p < 0.05$), Sections C-IDLE and B ($p < 0.01$), C-VIM and A ($p < 0.01$, including Bernoulli’s test) and Sections B and A ($p < 0.01$, including Bernoulli’s test).</p>			
<p>Amount of Time to Become Comfortable with “Other” Environment: A one-way ANOVA was conducted ($p < 0.01$); T-tests showed a significant difference between Sections C-VIM and B ($p < 0.01$) and Sections B and A ($p < 0.01$, including Bernoulli’s test).</p>			
<i>*Indicates two sections.</i>			

Table 66e: Section-by-Section Environment Usability Data – Third Survey (After Environment Switch) (CONT'D)

Section	Mishandling the “Other” Environment (Including IDLE)	Know How to Use the “Other” Environment (on a scale of 1 – 10)	Refer Back to VIM to Complete Tasks while using the “Other” Environment
A (N=13)	Absolutely Often - 0% Mostly Often - 0% Fairly Often - 0% 50/50 - 15% Slightly Often - 38% Mostly NOT Often - 31% Absolutely NOT Often - 8% No Response - 8%	(N=12**) Overall average: 8.50 **Provided a response	Not At All - 69% Mostly No - 15% Slightly Yes - 0% 50/50 - 8% Fairly Yes - 0% Mostly Yes - 0% Absolutely Yes - 0% No Response - 8%
C - IDLE (N=32)	Absolutely Often - 0% Mostly Often - 3% Fairly Often - 12% 50/50 - 18% Slightly Often - 15% Mostly NOT Often - 48% Absolutely NOT Often - 0% No Response - 3%	(N=28**) Overall average: 7.27 **Provided a response	Not At All - 42% Mostly No - 15% Slightly Yes - 6% 50/50 - 15% Fairly Yes - 12% Mostly Yes - 6% Absolutely Yes - 0% No Response - 3%
Section	Mishandling the “Other” Environment (Including VIM)	Know How to Use the “Other” Environment (on a scale of 1 – 10)	Refer Back to IDLE to Complete Tasks while using the “Other” Environment
B* (N=41)	Absolutely Often - 7% Mostly Often - 2% Fairly Often - 13% 50/50 - 22% Slightly Often - 20% Mostly NOT Often - 9% Absolutely NOT Often - 4% No Response - 22%	(N=26**) Overall average: 5.88 **Provided a response	Not At All - 13% Mostly No - 29% Slightly Yes - 0% 50/50 - 20% Fairly Yes - 9% Mostly Yes - 2% Absolutely Yes - 2% No Response - 24%
C - VIM (N=31)	Absolutely Often - 3% Mostly Often - 6% Fairly Often - 10% 50/50 - 13% Slightly Often - 19% Mostly NOT Often - 35% Absolutely NOT Often - 3% No Response - 10%	(N=25**) Overall average: 7.44 **Provided a response	Not At All - 23% Mostly No - 13% Slightly Yes - 6% 50/50 - 23% Fairly Yes - 6% Mostly Yes - 13% Absolutely Yes - 6% No Response - 10%
Statistical Significance			
<p>Mishandling the “Other” Environment: A one-way ANOVA was conducted ($p < 0.01$); T-tests showed a significant difference between Sections C-IDLE and B ($p < 0.05$) and Sections B and A ($p < 0.01$, including Bernoulli's test).</p> <p>Know How to Use the “Other” Environment: A one-way ANOVA was conducted ($p < 0.01$); T-tests showed a significant difference between Sections C-IDLE and A ($p < 0.01$, including Bernoulli's test), Sections C-VIM and B ($p < 0.05$), Sections C-IDLE and B ($p < 0.01$) and Sections B and A ($p < 0.01$, including Bernoulli's test).</p> <p>Refer Back to “Assigned” Environment to Complete Tasks while using the “Other” Environment: A one-way ANOVA was conducted ($p < 0.01$); T-tests showed a significant difference between Sections C-IDLE and A ($p < 0.01$, including Bernoulli's test), Sections C-VIM and A ($p < 0.01$, including Bernoulli's test) and Sections B and A ($p < 0.01$, including Bernoulli's test).</p>			
*Indicates two sections.			

Table 66f: Section-by-Section Environment Usability Data – Third Survey (After Environment Switch) (CONT'D)

Section	VIM vs. "Other" Environment (Which do you like more?)	Use "Other" Environment for Random Projects Outside of a Course	Use VIM for Random Projects Outside of a Course
A (N=13)	(N=12**) Assigned Environment - 8% Other Environment - 92% Neither/Doesn't Matter - 0% I Don't Know - 0% **Provided a response	(N=12**) Yes - 100% No - 0% I Don't Know/Maybe - 0% **Provided a response	(N=23**) Yes - 39% No - 57% I Don't Know/Maybe - 4% **Provided a response
C - IDLE (N=32)	(N=26**) Assigned Environment - 31% Other Environment - 59% Neither/Doesn't Matter - 10% I Don't Know - 0% **Provided a response	(N=27**) Yes - 74% No - 19% I Don't Know/Maybe - 7% **Provided a response	(N=25**) Yes - 64% No - 24% I Don't Know/Maybe - 12% **Provided a response
Section	IDLE vs. "Other" Environment (Which do you like more?)	Use "Other" Environment for Random Projects Outside of a Course	Use IDLE for Random Projects Outside of a Course
B* (N=41)	(N=29**) Assigned Environment - 35% Other Environment - 54% Neither/Doesn't Matter - 12% I Don't Know - 0% **Provided a response	(N=12**) Yes - 100% No - 0% I Don't Know/Maybe - 0% **Provided a response	(N=35**) Yes - 51% No - 29% I Don't Know/Maybe - 20% **Provided a response
C - VIM (N=31)	(N=27**) Assigned Environment - 48% Other Environment - 44% Neither/Doesn't Matter - 7% I Don't Know - 0% **Provided a response	(N=25**) Yes - 60% No - 32% I Don't Know/Maybe - 8% **Provided a response	(N=7**) Yes - 86% No - 14% I Don't Know/Maybe - 0% **Provided a response
Statistical Significance			
<p>"Assigned" Environment vs. "Other" Environment: One T-test showed a significant difference between Sections C-VIM and A ($p < 0.01$, including Bernoulli's test).</p> <p>Use "Other" Environment for Random Projects Outside of a Course: A one-way ANOVA was conducted ($p < 0.01$); T-tests showed a significant difference between Sections C-VIM and B ($p < 0.01$, including Bernoulli's test) and Sections C-VIM and A ($p < 0.01$, including Bernoulli's test), Sections C-IDLE and B ($p < 0.01$, including Bernoulli's test) and Sections C-IDLE and A ($p < 0.01$, including Bernoulli's test).</p> <p>Use "Assigned" Environment for Random Projects Outside of a Course: A one-way ANOVA was conducted ($p = 0.01$); T-tests showed a significant difference between Sections C-VIM and B ($p < 0.05$), Sections C-VIM and C-IDLE ($p < 0.05$) and Sections C-VIM and A ($p < 0.01$, including Bernoulli's test).</p>			
*Indicates two sections.			

6.6.4 Usability Survey Comparison²

This section discusses the differences in responses between the three usability assessments based on the students' feelings about these environments. This consisted of *comfort with the "assigned" environment, mishandling the "assigned" environment, confidence of doing another assignment with "assigned" environment, fondness of "assigned" environment, amount of time to become comfortable with "assigned" environment, and knowledge of using "assigned" environment*. Tables 67a – 67d displays these scores (as averages) along with statistical significances for each section.

Section A's comfort level with the assigned environments decrease significantly ($p < 0.01$, including *Bernoulli's test*) after switching from IDLE to VIM. When comparing their initial comfort levels for using IDLE (1st survey) and VIM (3rd survey), Section A showed a significantly higher comfort level for using IDLE ($p < 0.01$, including *Bernoulli's test*). Their initial level of confidence for doing another assignment with IDLE or VIM differed significantly ($p < 0.01$, including *Bernoulli's test*). There was also a significant change in their initial fondness for each environment. In particular, these students showed a higher fondness toward IDLE than VIM ($p < 0.01$, including *Bernoulli's test*).

Section B, who supposedly used VIM prior to IDLE, only showed significant changes between their initial comfort with using IDLE and VIM. In particular, these students showed a higher comfort with VIM than IDLE ($p < 0.05$). In Section C, students using IDLE after the switch showed a decrease in their comfort level ($p < 0.01$, including *Bernoulli's test*). Theoretically, the majority of Section C students used VIM prior to the switch. This was also true for students in Section C who were using VIM after the switch ($p < 0.05$). The IDLE students in

² Portions of this section are included in a paper that has been recently accepted for publication in the 56th Annual Human Factors and Ergonomics Society Conference.

Section C also showed decrease in their fondness for using this tool ($p < 0.01$, including Bernoulli's test).

Table 67a: CS 150 Demographics – Survey Comparison (Section A)

Comfort with “Assigned” Environment (average based on a 7 point Likert scale: <i>7 = absolutely comfortable; 1 = not comfortable at all</i>)	Mishandling the “Assigned” Environment (average based on a 7 point Likert scale: <i>7 = absolutely often; 1 = absolutely NOT often</i>)	Confident with Doing Another Assignment with “Assigned” Environment (average based on a 7 point Likert scale: <i>7 = absolutely confident; 1 = not confident at all</i>)
(N=31*) 1 st survey - 4.51	(N=31*) 1 st survey - 4.10	(N=31*) 1 st survey - 4.61
(N=26*) 2 nd survey - 5.08	(N=26*) 2 nd survey - 4.42	(N=26*) 2 nd survey - 4.54
(N=13*) **3 rd survey - 3.08	(N=13*) **3 rd survey - 3.08	(N=13*) **3 rd survey - 2.69
Fondness of “Assigned” Environment (average based on a 7 point Likert scale: <i>7 = absolutely like; 1 = not at all</i>)	Amount of Time to Become Comfortable with “Assigned” Environment (average based on a 6 point Likert scale: <i>6 = 1 week or less; 1 = still not comfortable</i>)	Know How to Use “Assigned” Environment (on a scale of 1 – 10: 10 = very well; 1 = not so well)
(N=31*) 1 st survey - 4.55	(N=23*) 2 nd survey - 3.70	(N=23*) 2 nd survey - 6.23
(N=26*) 2 nd survey - 4.46	(N=13*) **3 rd survey - 2.23	(N=12*) **3 rd survey - 3.92
(N=13*) **3 rd survey - 2.31		
Statistical Significance		
Comfort with “Assigned” Environment: A one-way ANOVA was conducted ($p < 0.01$); T-Tests showed a significant difference between the 1 st and 3 rd surveys ($p < 0.01$, including Bernoulli’s test) and the 2 nd and 3 rd surveys ($p < 0.01$, including Bernoulli’s test).		
Confident with Doing Another Assignment with “Assigned” Environment: A one-way ANOVA was conducted ($p < 0.01$); T-Tests showed a significant difference between the 1 st and 3 rd surveys ($p < 0.01$, including Bernoulli’s test) and the 2 nd and 3 rd surveys ($p < 0.01$, including Bernoulli’s test).		
Fondness of “Assigned” Environment: A one-way ANOVA was conducted ($p < 0.01$); T-Tests showed a significant difference between the 1 st and 3 rd surveys ($p < 0.01$, including Bernoulli’s test) and the 2 nd and 3 rd surveys ($p < 0.01$, including Bernoulli’s test).		
Know How to Use “Assigned” Environment: T-Test showed a significant difference between the 2 nd and 3 rd surveys ($p < 0.01$, including Bernoulli’s test).		
<i>*Provided a Response; **After Environment Switch</i>		

Table 67b: CS 150 Demographics – Survey Comparison (Section B)

Comfort with “Assigned” Environment (average based on a 7 point Likert scale: <i>7 = absolutely comfortable; 1 = not comfortable at all</i>)		Mishandling the “Assigned” Environment (average based on a 7 point Likert scale: <i>7 = absolutely often; 1 = absolutely NOT often</i>)		Confident with Doing Another Assignment with “Assigned” Environment (average based on a 7 point Likert scale: <i>7 = absolutely confident; 1 = not confident at all</i>)	
(N=29*)	1 st survey - 4.79	(N=29*)	1 st survey - 3.90	(N=29*)	1 st survey - 4.45
(N=49*)	2 nd survey - 5.22	(N=49*)	2 nd survey - 4.22	(N=49*)	2 nd survey - 4.88
(N=41*)	**3 rd survey - 4.07	(N=39*)	**3 rd survey - 4.41	(N=40*)	**3 rd survey - 4.25
Fondness of “Assigned” Environment (average based on a 7 point Likert scale: <i>7 = absolutely like; 1 = not at all</i>)		Amount of Time to Become Comfortable with “Assigned” Environment (average based on a 6 point Likert scale: <i>6 = 1 week or less; 1 = still not comfortable</i>)		Know How to Use “Assigned” Environment (on a scale of 1 – 10: <i>10 = very well; 1 = not so well</i>)	
(N=29*)	1 st survey - 4.69	(N=49*)	2 nd survey - 4.18	(N=45*)	2 nd survey - 6.92
(N=49*)	2 nd survey - 4.84	(N=39*)	**3 rd survey - 3.64	(N=39*)	**3 rd survey - 5.15
(N=40*)	**3 rd survey - 4.15				
Statistical Significance					
<p>Comfort with “Assigned” Environment: A one-way ANOVA was conducted ($p < 0.01$); T-Tests showed a significant difference between the 1st and 3rd surveys ($p < 0.05$) and the 2nd and 3rd surveys ($p < 0.01$).</p> <p>Know How to Use “Assigned” Environment: T-Test showed a significant difference between the 2nd and 3rd surveys ($p < 0.01$).</p>					
<i>*Provided a Response; **After Environment Switch</i>					

Table 67c: CS 150 Demographics – Survey Comparison (Section C)

IDLE					
Comfort with “Assigned” Environment (average based on a 7 point Likert scale: 7 = <i>absolutely comfortable</i> ; 1 = <i>not comfortable at all</i>)		Mishandling the “Assigned” Environment (average based on a 7 point Likert scale: 7 = <i>absolutely often</i> ; 1 = <i>absolutely NOT often</i>)		Confident with Doing Another Assignment with “Assigned” Environment (average based on a 7 point Likert scale: 7 = <i>absolutely confident</i> ; 1 = <i>not confident at all</i>)	
(N=13*)	1 st survey - 5.69	(N=13*)	1 st survey - 5.38	(N=13*)	1 st survey - 5.92
(N=7*)	2 nd survey - 6.42	(N=7*)	2 nd survey - 5.14	(N=7*)	2 nd survey - 5.14
(N=32*)	**3 rd survey - 5.31	(N=32*)	**3 rd survey - 5.19	(N=32*)	**3 rd survey - 5.22
Fondness of “Assigned” Environment (average based on a 7 point Likert scale: 7 = <i>absolutely like</i> ; 1 = <i>not at all</i>)		Amount of Time to Become Comfortable with “Assigned” Environment (average based on a 6 point Likert scale: 6 = <i>1 week or less</i> ; 1 = <i>still not comfortable</i>)		Know How to Use “Assigned” Environment (on a scale of 1 – 10: 10 = <i>very well</i> ; 1 = <i>not so well</i>)	
(N=13*)	1 st survey - 5.31	(N=7*)	2 nd survey - 4.86	(N=6*)	2 nd survey - 7.33
(N=7*)	2 nd survey - 5.71	(N=30*)	**3 rd survey - 4.30	(N=26*)	**3 rd survey - 6.87
(N=32*)	**3 rd survey - 4.06				
VIM					
Comfort with “Assigned” Environment (average based on a 7 point Likert scale: 7 = <i>absolutely comfortable</i> ; 1 = <i>not comfortable at all</i>)		Mishandling the “Assigned” Environment (average based on a 7 point Likert scale: 7 = <i>absolutely often</i> ; 1 = <i>absolutely NOT often</i>)		Confident with Doing Another Assignment with “Assigned” Environment (average based on a 7 point Likert scale: 7 = <i>absolutely confident</i> ; 1 = <i>not confident at all</i>)	
(N=27*)	1 st survey - 5.74	(N=27*)	1 st survey - 4.74	(N=27*)	1 st survey - 5.85
(N=30*)	2 nd survey - 6.40	(N=30*)	2 nd survey - 5.17	(N=30*)	2 nd survey - 6.00
(N=31*)	**3 rd survey - 5.84	(N=31*)	**3 rd survey - 5.16	(N=31*)	**3 rd survey - 5.97
Fondness of “Assigned” Environment (average based on a 7 point Likert scale: 7 = <i>absolutely like</i> ; 1 = <i>not at all</i>)		Amount of Time to Become Comfortable with “Assigned” Environment (average based on a 6 point Likert scale: 6 = <i>1 week or less</i> ; 1 = <i>still not comfortable</i>)		Know How to Use “Assigned” Environment (on a scale of 1 – 10: 10 = <i>very well</i> ; 1 = <i>not so well</i>)	
(N=27*)	1 st survey - 5.89	(N=28*)	2 nd survey - 4.54	(N=29*)	2 nd survey - 7.62
(N=30*)	2 nd survey - 5.67	(N=29*)	**3 rd survey - 4.41	(N=26*)	**3 rd survey - 7.21
(N=31*)	**3 rd survey - 5.26				
Statistical Significance					
<p>Comfort with “Assigned” Environment (IDLE): T-Test showed a significant difference between the 2nd and 3rd surveys ($p < 0.01$, including Bernoulli’s test).</p> <p>Comfort with “Assigned” Environment (VIM): T-Test showed a significant difference between the 1st and 2nd surveys ($p < 0.05$) and 2nd and 3rd surveys ($p < 0.05$).</p> <p>Fondness of “Assigned” Environment (IDLE): A one-way ANOVA was conducted ($p < 0.05$); T-Test showed a significant difference between the 2nd and 3rd surveys ($p < 0.01$, including Bernoulli’s test).</p>					
*Provided a Response; **After Environment Switch					

6.7 Discussion

For some measures, Section C showed a stronger performance than their counterparts in sections A and B. Throughout the semester, Section C was less intimidated with programming and computer science. This section also had a higher self-efficacy for programming while providing higher scores about their environment's usability. For each exam, Section C received a higher proficiency rating.

Sections A and B's performance were similar in some cases. For example, their proficiency ratings were relatively close on each exam. Their response averages from the usability questions were also insignificantly different prior to the environment switch. There were cases however where both sections showed contrasting results. For example, Section A showed a higher self-efficacy for programming than Section B during the final assessment, but showed a higher tendency to struggle with using VIM (than Section B did with IDLE) after switching environments. Section A also showed a higher preference for IDLE (than VIM) on the usability surveys. During the final usability assessment, Section B reported higher scores about using IDLE (than Section A did with VIM). In addition, Section B students (who used VIM) performed better than Section A (who used IDLE) during the protocol analysis. Section B also showed a higher representation of students continuing to use their new environment (IDLE) than Section A (VIM) after switching environments.

The results from Pennington's Model indicated minor differences between the three sections as well as their respective versions of the survey. One difference occurred between

sections B and C for understanding *Control Flow* during the first survey. Another difference occurred when comparing the two versions of the survey for *Control Flow* (1st survey) and *Program State* (2nd survey). However, the majority of students were able to answer each question correctly regardless of their version of the survey, section in the course, or environment used or preferred.

The first programming procedures survey showed a higher percentage of VIM users who provided the correct response (in its entirety) for *understanding compilation* than those using IDLE. On the second programming procedures survey, IDLE users showed a higher percentage for *explaining the process of creating a program*. However, this could have been due to the low response rate from Section B during this particular assessment. On both surveys, understanding how to *link* a program consistently received the lowest percentage of correct responses.

6.8 Summary

In this study, the objective was to measure any difference in impact between moderately and low assistive environments on novice programmers (as in the CS1 – Laboratory Study). Based on the results from this study, it can be concluded that moderately and low assistive environments have potential advantages and disadvantages for teaching novices how to program. Based on IDLE's feature set, this environment has the potential for instant use by programmers without major drawbacks. In comparison to VIM, IDLE was shown to have a lower learning curve and a lighter usability load during the protocol analysis and the final usability assessment. However, this study also revealed that IDLE possibly prevented students from developing a more accurate mental model for programming, in addition to making an easier transition to VIM. This was found to be the case during the protocol analysis, programming procedures assessment (*understanding compilation*), and Section A's final usability assessment. IDLE's potential disadvantages however can complement the advantages for using VIM (and vice versa). For example, the protocol analysis showed VIM to potentially have a higher learning curve than IDLE. However, students who used originally used VIM during the semester tended to make an easier transition to IDLE.

This study also revealed that students may have preferred to use their original environment over a new one regardless of its easiness or complexity of use. This was the case for the majority of students who participated in the protocol analysis as well as those who responded to the final usability assessment. This preference can also be linked to section A and B's responses on the first two usability assessments, which showed no contrasting issues with using

IDLE or VIM respectively.

Another indicator is the proficiency ratings between IDLE and VIM users. There were no significant contrasts between the proficiency ratings for these two environments. Even though these environments may have contrasting learning curves, these students had to learn enough about their respective environment in order to complete assignments and exams during the semester, especially prior to the environment switch. A question remains of the actual moment a novice becomes competent with using either a moderately assistive or low assistive environment. With exception to the Section C (the honor section), this study showed no direct contrast regarding the amount of time for students to learn how to use IDLE or VIM respectively.

7. THREATS TO VALIDITY

There are potential threats that could affect the validity of the results and conclusions that appear from this research. Each potential threat is detailed below:

- ***Evaluating only a select set of programming environments during this research.*** Every existing programming environment was not evaluated during this research. Theories, prior conclusions, and anecdotal evidence about certain programming environments were used a point of reference for conducting this research. The objective was to conduct studies while applying measures that would either support or reject some of these prior findings or beliefs.
- ***Short-term durations for the Aliceville Outreach and CS1-Laboratory Study.*** The Aliceville Outreach and CS1-Laboratory Study provided preliminary results and conclusions for this research. However, both studies only considered the student's behavior for a short-term duration; Aliceville Outreach (5 weeks) and CS1-Laboratory Study (1 day). Both studies also involved students who did not (or would not) pursue computer science as an area of interest. However, a semester-long study (like the CS1 study) possibly controlled for these and related factors.
- ***Incommensurable comparisons between certain sections of a course.*** Section C tended to show a stronger performance during each measure of the CS1 study. Their behavior introduced the potential for making "apples to oranges" comparisons between section C and sections A and B respectively. A related factor was the presence of different instructors in this course. Each section had a different instructor, which introduces the possibility of varying teaching styles for each section.

- ***Using environments contrary to the one assigned.*** During the CS1 study, some students chose to use an environment contrary to the one assigned in their respective sections. One reason was that CS150 and 250 students traditionally use VIM to learn programming. During this study, some students used VIM due to the belief that it would be required for CS250, while others knew of acquaintances who took CS150 prior to the Fall 2011 semester. There were also cases where students preferred to use environments other than IDLE or VIM (ex. GEDIT) after a subsequent amount of exposure to the Linux platform.
- ***Format of the CS1 course and study (influenced low sample representations).*** There were students who either stop attending class or drop the CS1 course as the semester progressed. There were also students who became agitated with participating in this study because of the various assessments that were administered. It is possible that the format of the CS1 course and the semester-long study influenced low sample representations and lack of responses respectively on some of these assessments, especially during the final assessments of the semester.

8. FUTURE WORK

There is additional work that can be done in regards to this research. One is to study students as they matriculate through a CS curriculum. This approach may be able to provide a more detailed understanding of how a student's mental model for programming is acquired or modified, along with the environment(s) that are being used during this process.

Another future work is to adjust the instruments employed during a study to obtain a high number of responses at a consistent level. As previously mentioned, some students involved in the CS1 study became agitated with undergoing various assessments at different points during the semester. This may have resulted in lower response rates for some of the assessments. A related future work is to assess students at particular times of the semester where the attendance rate tends to be high on a consistent basis.

Another area of future work relates to the actual programming environments. Some of the environments used during the CS1-Laboratory and CS1 studies consisted of tools primarily for Python programming. A primary future work is to apply evaluations to environments outside of Python. Another future plan is to find an accurate approach to control for students who use environments other than the "assigned" ones during a study. An additional work is to continue further studies on novices who are exposed to programming through low assistive environments with the objective of determining the actual moment they acquire the understanding (or mental model) for using such tools. This question also introduces another future work for determining whether a learning curve trend exists for the feature set continuum.

Another future work is to determine whether novices necessarily need exposure to command line programming while matriculating through a CS curriculum. Prior studies have used moderately and highly assistive environments as a way to attract students with the intention of retaining them as CS majors. However, the question remains of whether students, who initially learn to program by using highly or moderately assistive environments, may eventually need exposure to a low assistive environment in order round out their skill sets and possibly enhance their mental model for programming.

A related future work is to measure the transition of students to command line programming after prior exposure to either a visual or command line environment. The CS1-Laboratory study showed that students continued to struggle with using Notepad even though many of them had prior exposure to VIM. The CS1-Study showed that students who originally used IDLE struggled when transitioning VIM. The question remains of whether prior exposure to either a visual or command line environment would influence the learning curve for using a new (or different) low assistive environment.

9. CONCLUSION

The overall objective of this research was to determine whether certain programming environments are potentially more appropriate for teaching novices how to program. The literature review discussed debates about appropriate paradigms and languages to use for introducing a novice to the concept of programming. Related studies have evaluated the effect of different programming environments on novices. However, majority of these studies only evaluated visual environments (both moderately and highly assistive).

This research evaluated different programming environments with varying feature sets through measures of engagement, comprehension, efficiency and usability. These measures were applied to three studies (*Aliceville Outreach*, *CS1-Laboratory Study*, and *CS1 Study*). The Aliceville outreach showed that many of the students were comfortable with using PREOP, understood most of the programming concepts, and showed a slightly above average self-efficacy for programming. However, the main objective for this study was to test the validity of these applied measures for evaluating programming environments. The CS1-Laboratory Study showed that students struggled with using a less assistive environment (Notepad) regardless of their persistence with programming, but were able to use moderately assistive environments (IDLE and PyScripter) more effectively. The results from the CS1 Study indicated that moderately and low assistive environments present potential advantages and disadvantages for novices when learning to program. In particular, IDLE provided the students with a lower learning curve than VIM (less assistive environment). On the other hand, VIM may

have equipped its users with a better mental model for understanding the underlying factors of programming while enabling them to make easier transitions to using IDLE after switching environments.

To give an official conclusion to this research, the proposed hypotheses are addressed. As mentioned in Chapter 1, the sub-hypotheses will either reject or not reject the alternative hypothesis that *a moderately assistive environment is more effective for teaching novices how to program than a low assistive environment* or cause for the null hypothesis, *a moderately assistive environment is NOT more effective for teaching novices how to program than a low assistive environment*, to not be rejected.

For sub-hypothesis Ha_1 , *a moderately assistive environment is more engaging*, the results showed that students who had prior programming experience and/or were in an advanced section tended to have a higher self-efficacy for programming. This former result was found to be the case in the CS1-Laboratory Study while the latter result was true during the CS1-Study. Therefore, sub-hypothesis Ha_1 can be rejected.

For sub-hypothesis Ha_2 , *a moderately assistive environment helps programmers better understand the concepts and procedures of programming*, the results from the CS1 study (1st Programming Procedures survey) indicated that students using IDLE scored significantly lower for *understanding compilation*. Also, the protocol analysis showed that students who originally used IDLE struggled with understanding the programming procedures for VIM. Therefore, sub-hypothesis Ha_2 can be rejected.

For sub-hypothesis Ha_3 , *a moderately assistive environment is more efficient*, the results from the CS1-Laboratory Study showed this to be true initially. The IDLE students, who had less prior programming experience, completed their task quicker than those using Notepad. The

potential difference in the learning curves between moderately and low assistive environments may initially influence a difference in efficiency. However, the efficiency for using an environment could increase as a novice becomes more acclimated to the tool. For example, the students in the CS1-Study had to learn enough about their respective environment in order to complete assignments and exams during the semester. In addition, sections A and B, who primarily used IDLE and VIM respectively, showed insignificant differences for their proficiency ratings on each exam. A possible reason for this insignificance may be linked to the fact that both sections respectively became accustomed to using their environment prior to the first efficiency assessment. Therefore, sub-hypothesis Ha_3 can be rejected.

For sub-hypothesis Ha_4 , *a moderately assistive environment has better usability*, the results from the CS1-Laboratory Study showed this to be initially true when focusing on the IDLE students. Even though they had less prior programming experience, the IDLE students were able to complete their tasks more effectively than those using Notepad. The CS1-Laboratory Study also supports this sub-hypothesis when considering a student's programming behavior after initial exposure to a moderately assistive environment. For example, the PyScripter group, who had prior experience with moderately assistive environments, gave significantly higher scores for some questions about their environment's usability than those using Notepad, who had prior experience low assistive environments. The CS1-Study showed that Section A, in many cases, provided significantly higher responses about using IDLE than for VIM. After switching environments, Section B provided significantly higher responses for some questions about using IDLE than Section A did with VIM. The protocol analysis also showed that most of the students who originally used VIM were able to learn enough about IDLE to use it effectively. Therefore, sub-hypothesis Ha_4 cannot be rejected.

By rejecting three of the four sub-hypotheses, the null hypothesis for this research is not rejected. The results from the research however show that both moderately and low assistive environments provide potential benefits for novices. A moderately assistive environment is able to impose a lower learning curve while a low assistive environment can potentially help a novice acquire a more helpful mental model for understanding the underlying concepts of programming.

BIBLIOGRAPHY

- [1] Adelson, B. 1984. When novices surpass experts: the difficulty of a task may increase with expertise. *Journal of Experimental Psychology: Learning, Memory, and Cognition*. Vol. 10, (3), 1984. 483 – 495. Reprinted in *Human Factors in Software Development* (2nd ed.); Bill Curtis (ed.). Washington, DC. IEEE Computer Society Press, 1985. 55 – 67.
- [2] *Alice*. <http://www.alice.org/> (accessed October 9, 2009).
- [3] Allen, E., Cartwright, R., and Stoler, B. 2002. DrJava: a lightweight pedagogic environment for Java. In *Proceedings of the 33rd SIGCSE Technical Symposium on Computer Science Education* (Cincinnati, Kentucky, February 27 - March 03, 2002). SIGCSE '02. ACM, New York, NY, 137-141.
- [4] Astrachan, O., Bruce, K., Koffman, E., Kölling, M., and Reges, S. 2005. Resolved: objects early has failed. *SIGCSE Bull.* 37, 1 (Feb. 2005), 451-452.
- [5] Bailie, F., Courtney, M., Murray, K., Schiaffino, R., and Tuohy, S. 2003. Objects first - does it work?. *J. Comput. Small Coll.* 19, 2 (Dec. 2003), 303-305.
- [6] Barnes, D. J. 2002. Teaching introductory Java through LEGO MINDSTORMS models. In *Proceedings of the 33rd SIGCSE Technical Symposium on Computer Science Education* (Cincinnati, Kentucky, February 27 - March 03, 2002). SIGCSE '02. ACM, New York, NY, 147-151.
- [7] Baum, Dave. *Definitive Guide to LEGO MINDSTORMS*. Academic Press, 2000.
- [8] Beaubouef, T. and Mason, J. 2005. Why the High Attrition Rate for Computer Science Students: Some Thoughts and Observations. *Inroads* 37, 2.
- [9] Becker, K. 2002. Back to Pascal: retro but not backwards. *J. Comput. Small Coll.* 18, 2 (Dec. 2002), 17-27.
- [10] Bednarik, R. and Tukiainen, M. 2005. Effects of display blurring on the behavior of novices and experts during program debugging. In *CHI '05 Extended Abstracts on Human Factors in Computing Systems* (Portland, OR, USA, April 02 - 07, 2005). CHI '05. ACM, New York, NY, 1204-1207.

- [11] Bednarik, R. and Tukiainen, M. 2004. Visual attention tracking during program debugging. In *Proceedings of the Third Nordic Conference on Human-Computer interaction* (Tampere, Finland, October 23 - 27, 2004). NordiCHI '04, vol. 82. ACM, New York, NY, 331-334.
- [12] Ben-Ari, Mordechai (Moti). 2008. "The Effect Of The Jeliot Animation System On Learning Elementary Programming." 20-30.
- [13] Ben-Ari, Mordechai (Moti). 2001. Constructivism in computer science education. *J. Comput. Math. Sci. Teach.* 20, 1 (Jan. 2001), 45-73.
- [14] Bergin S., Reilly R. 2005. The Influence of Motivation and Comfort-level on Learning to Program. In *Proceedings of the 17th Workshop on Psychology of Programming*, PPIG'05.
- [15] Biddle, R. and Tempero, E. 1998. Java pitfalls for beginners. *SIGCSE Bull.* 30, 2 (Jun. 1998), 48-52.
- [16] Blank, D.S., Kumar, D., Meeden, L., and Yanco, H. 2006. The Pyro toolkit for AI and robotics. *AI Magazine*. Vol 27(1), Spring 2006. AAAI Press.
- [17] Blaszczyk, M. *Professional MFC With Visual C++ 6*. Birmingham UK: Wrox Press Ltd, 1999.
- [18] Bloom, B. S., Krathwohl, D. R. & Masia, B. B. 1956. Taxonomy of educational objectives: the classification of educational goals. Handbook I: Cognitive domain.
- [19] *BlueJ - the interactive Java environment*. <http://www.bluej.org/> (accessed 9 2009, October).
- [20] Bolton, David. "SlickEdit - Programmer's Editor." *About.com*. 2009. <http://cplus.about.com/od/softwarereviews/fr/slickedit.htm> (accessed December 15, 2009).
- [21] Brown, R., Davis, J., Rebelsky, S. A., and Harvey, B. 2009. Whither scheme?: 21st century approaches to scheme in CS1. In *Proceedings of the 40th ACM Technical Symposium on Computer Science Education* (Chattanooga, TN, USA, March 04 - 07, 2009). SIGCSE '09. ACM, New York, NY, 551-552.
- [22] Bruce, K. B. 2005. Controversy on how to teach CS 1: a discussion on the SIGCSE-members mailing list. *SIGCSE Bull.* 37, 2 (Jun. 2005), 111-117.
- [23] Bruno, V., & Al-Qaimari, G. 2004. Usability Attributes: An Initial Step Toward Effective User-Centred Development. OZCHI, Wollongong, Australia.

- [24] Card, Stuart, Thomas P Moran, and Allen Newell. *The Psychology of Human Computer Interaction*. Hillsdale, New Jersey: Lawrence Erlbaum Associates, 1983.
- [25] Carlisle, M. C., Wilson, T. A., Humphries, J. W., and Hadfield, S. M. 2005. RAPTOR: a visual programming environment for teaching algorithmic problem solving. *SIGCSE Bull.* 37, 1 (Feb. 2005), 176-180.
- [26] Chen, Z. and Marx, D. 2005. Experiences with Eclipse IDE in programming courses. *J. Comput. Small Coll.* 21, 2 (Dec. 2005), 104-112.
- [27] Clark, D., MacNish, C., and Royle, G. F. 1998. Java as a teaching language—opportunities, pitfalls and solutions. In *Proceedings of the 3rd Australasian Conference on Computer Science Education* (The University of Queensland, Australia, July 08 - 10, 1998). ACSE '98, vol. 3. ACM, New York, NY, 173-179.
- [28] Close, R., Kopec, D., and Aman, J. 2000. CS1: perspectives on programming languages and the breadth-first approach. *J. Comput. Small Coll.* 15, 5 (May. 2000), 228-234.
- [29] Coleman, D., Confino, J., Koletzke, P., McCallister, B., Purcell, T., and Shepard, J. 2004 *Java™ IDE Shootout*. Powerpoint Presentation, 2004 JavaOne Conference.
- [30] Conway, M., Pausch, R., Gossweiler, R., and Burnette, T. 1994. Alice: a rapid prototyping system for building virtual environments. In *Conference Companion on Human Factors in Computing Systems* (Boston, Massachusetts, United States, April 24 - 28, 1994). C. Plaisant, Ed. CHI '94. ACM, New York, NY, 295-296.
- [31] Cook, C. R. and Gellenbeck, E. M. 1991. *An Investigation of Procedure and Variable Names as Beacons During Program Comprehension*. Technical Report. UMI Order Number: 91-60-02., Oregon State University.
- [32] Cooper, S., Moskal, B., and Lurie, D., 2004. Evaluating the effectiveness of a new instructional approach. In *Proceedings of the 35th SIGCSE Technical Symposium on Computer Science Education* (Norfolk, Virginia, USA, March 03 - 07, 2004). SIGCSE '04. ACM, New York, NY, 75-79.
- [33] Cooper, S., Dann, W., and Pausch, R. 2003. Teaching objects-first in introductory computer science. In *Proceedings of the 34th SIGCSE Technical Symposium on Computer Science Education* (Reno, Nevada, USA, February 19 - 23, 2003). SIGCSE '03. ACM, New York, NY, 191-195.
- [34] Corritore, C. L. and Wiedenbeck, S. 1991. What do novices learn during program comprehension? *International Journal of Human-Computer Interaction*, 3(2), 199- 222.
- [35] Crosby, M. E. and Stelovsky, J. 1990. How Do We Read Algorithms? A Case Study. *Computer* 23, 1 (Jan. 1990), 24-35.

- [36] Cross, J. H. and Hendrix, T. D. 2007. JGRASP: an integrated development environment with visualizations for teaching Java in CS1, CS2, and beyond. *J. Comput. Small Coll.* 23, 1 (Oct. 2007), 5-7.
- [37] Cutrell, E. and Guan, Z. 2007. An eye-tracking study of information usage in Web search: Variations in target position and contextual snippet length. In *Proc. CHI 2007*. ACM Press (2007).
- [38] Dale, N. 2005. Content and emphasis in CS1. *SIGCSE Bull.* 37, 4 (Dec. 2005), 69-73.
- [39] Davis, S. 1990. Plans, goals, and selection rules in the comprehension of computer programs. *Behaviour and Information Technology.* 9, 3, 201 – 214.
- [40] Decker, R. and Hirshfield, S. 1994. The top 10 reasons why object-oriented programming can't be taught in CS 1. In *Proceedings of the Twenty-Fifth SIGCSE Symposium on Computer Science Education* (Phoenix, Arizona, United States, March 10 - 12, 1994). SIGCSE '94. ACM, New York, NY, 51-55.
- [41] Depasquale, P. J. 2003. *Implications on the Learning of Programming Through the Implementation of Subsets in Program Development Environments*. Doctoral Thesis. UMI Order Number: AAI3095195., Virginia Polytechnic Institute and State University.
- [42] Dillon E., Anderson M., and Brown M. 2012. Comparing Feature Assistance Between Programming Environments and Their *Effect* on Novice Programmers. *Journal for Computing Sciences in Colleges*, 27, 5.
- [43] Dillon E., Anderson M., and Brown M. 2012. Comparing Mental Models of Novice Programmers when using Visual and Command Line Environments. In *Proceedings of the 50th Annual ACM Southeast Conference*.
- [44] Dodds, Z., Alvarado, C., Kuenning, G., and Libeskind-Hadas, R. 2007. Breadth-first CS 1 for scientists. *SIGCSE Bull.* 39, 3 (Jun. 2007), 23-27.
- [45] *DrJava*. <http://www.drjava.org/> (accessed October 9, 2009).
- [46] Duchowski, Andrew T. *Eye Tracking Methodology: Theory and Practice*. London: Springer, 2003.
- [47] Duchowski, Andrew T. *Eye Tracking Methodology: Theory and Practice*. London: Springer, 2007.
- [48] Duchowski, A. 2002. A breadth-first survey of eye-tracking applications. *Behavior Research Methods, Instruments and Computers* 34, 4, 455--470.
- [49] *Eclipse*. <http://www.eclipse.org/> (accessed October 15, 2009).

- [50] Erwin, Benjamin. *Creative Projects with LEGO(R) Mindstorms(TM)*. Upper Saddle River, NJ: Addison-Wesley, 2001.
- [51] Feldman, M. B. 1992. Ada experience in the undergraduate curriculum. *Commun. ACM* 35, 11 (Nov. 1992), 53-67.
- [52] Fenwick, J. B., Norris, C., Barry, F. E., Rountree, J., Spicer, C. J., and Cheek, S. D. 2009. Another look at the behaviors of novice programmers. In *Proceedings of the 40th ACM Technical Symposium on Computer Science Education* (Chattanooga, TN, USA, March 04 - 07, 2009). SIGCSE '09. ACM, New York, NY, 296-300.
- [53] Findler, R. B., Flanagan, C., Flatt, M., Krishnamurthi, S., and Felleisen, M. 1997. DrScheme: A Pedagogic Programming Environment for Scheme. In *Proceedings of the 9th international Symposium on Programming Languages: Implementations, Logics, and Programs: including A Special Trach on Declarative Programming Languages in Education* (September 03 - 05, 1997). H. Glaser, P.H. Hartel, and H. Kuchen, Eds. Lecture Notes In Computer Science, vol. 1292. Springer-Verlag, London, 369-388.
- [54] Gellenbeck, E. and Cook, C. An Investigation of Procedure and Variable Names as Beacons During Program Comprehension. *Proceedings of the 4th Workshop of Empirical Studies of Programmers* 1991; 65 – 81.
- [55] Goldwasser, M. H. and Letscher, D. 2008. Teaching an object-oriented CS1 -: with Python. In *Proceedings of the 13th Annual Conference on innovation and Technology in Computer Science Education* (Madrid, Spain, June 30 - July 02, 2008). ITiCSE '08. ACM, New York, NY, 42-46.
- [56] Good, J. Programming Paradigms, Information Types and Graphical Representations: Empirical Investigations of Novice Program Comprehension. PhD thesis, University of Edinburgh, 1999.
- [57] GNU/LINUX. *Anjuta DevStudio: GNOME Integrated Development Environment*. <http://projects.gnome.org/anjuta/index.shtml> (accessed December 15, 2009).
- [58] Gressard, B.H. Loyd and C. "Reliability and Factorial Validity of Computer Attitude Scales." *Educational and Psychological Measurement*, 1984: 401-505.
- [59] Gugerty, L. and Olson, G. M. 1986. Comprehension differences in debugging by skilled and novice programmers. In *Papers Presented At the First Workshop on Empirical Studies of Programmers on Empirical Studies of Programmers* (Washington, D.C., United States). E. Soloway and S. Iyengar, Eds. Ablex Publishing Corp., Norwood, NJ, 13-27.

- [60] Guzdial, M. 2004. Programming environments for novices. In *Computer Science Education Research*. S. Fincher and M. Petre (Eds.). Swets and Zeitlinger. Chapter 3.
- [61] Hadjerrouit, S. 1998. Java as first programming language: a critical evaluation. *SIGCSE Bull.* 30, 2 (Jun. 1998), 43-47.
- [62] Hagan, D., and Markham, S. 2000. Teaching Java with the BlueJ environment. In *Ascilite*, 2000.
- [63] Hessling, Mark. "The Hessling Editor." 2006. <http://hessling-editor.sourceforge.net/> (accessed December 15, 2009).
- [64] Hickey, T. J. 2004. Scheme-based web programming as a basis for a CS0 curriculum. In *Proceedings of the 35th SIGCSE Technical Symposium on Computer Science Education* (Norfolk, Virginia, USA, March 03 - 07, 2004). SIGCSE '04. ACM, New York, NY, 353-357.
- [65] Holt, R. W., Boehm-Davis, D. A., and Shultz, A. C. 1987. Mental representations of programs for student and professional programmers. In *Empirical Studies of Programmers: Second Workshop*, G. M. Olson, S. Sheppard, and E. Soloway, Eds. Ablex Series Of Monographs, Edited Volumes, And Texts. Ablex Publishing Corp., Norwood, NJ, 33-46.
- [66] Hughes, C. and Buckley, J. A Framework for Evaluating Comprehension of Concurrent Software. In *Proceedings of the 14th Workshop of the Psychology of Programming Interest Group*. Carlow IT. 2004.
- [67] *IDM Computing Solutions, Inc.* <http://www.ultraedit.com/index.html> (accessed December 18, 2009).
- [68] IntelliJ. *The Most Intelligent Java IDE*. 2009. <http://www.jetbrains.com/idea/> (accessed November 16, 2009).
- [69] *Introduction to SEDIT: XEDIT and PDF with a GUI*. <http://www.sedit.com/xeditgui.html> (accessed December 15, 2009).
- [70] Jacob, R. J. 1990. What you look at is what you get: eye movement-based interaction techniques. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems: Empowering People* (Seattle, Washington, United States, April 01 - 05, 1990). J. C. Chew and J. Whiteside, Eds. CHI '90. ACM, New York, NY, 11-18.
- [71] Jadud, M. and Henriksen, P. 2009. Flexible, reusable tools for studying novice programmers. In *Proceedings of the Fifth international Workshop on Computing Education Research Workshop* (Berkeley, CA, USA, August 10 - 11, 2009). ICER '09. ACM, New York, NY, 37-42.

- [72] Jadud, M. 2006. Methods and tools for exploring novice compilation behaviour. In *Proceedings of the Second international Workshop on Computing Education Research* (Canterbury, United Kingdom, September 09 - 10, 2006). ICER '06. ACM, New York, NY, 73-84.
- [73] Jadud, M. A first look at novice compilation behavior using bluej. In *16th Annual Workshop of the Psychology of Programming Interest Group (PPIG 2004)*, Institute of Technology, Carlow, Ireland, April 2004.
- [74] Jansen, A.R., Blackwell, A.F. and Marriott, K. A tool for tracking visual attention: The Restricted Focus Viewer. *Behavior Research Methods, Instruments, and Computers*, 35(1), 57--69, 2003.
- [75] *JBuilder*. 2009. <http://www.embarcadero.com/products/jbuilder> (accessed November 15, 2009).
- [76] *JCreator*. 2009. <http://www.jcreator.com/> (accessed November 15, 2009).
- [77] Jeffries, R. "A comparison of the debugging behavior of expert and novice programmers." *American Educational Research Association*. New York, 1982.
- [78] *JGrasp: An Integrated Development Environment with Visualizations for Improving Software Comprehensibility*. 2009. <http://www.jgrasp.org/> (accessed November 15, 2009).
- [79] Joachims, T., Granka, L., Pan, B., Hembrooke, H., and Gay, G. 2005. Accurately interpreting clickthrough data as implicit feedback. In *Proceedings of the 28th Annual international ACM SIGIR Conference on Research and Development in information Retrieval* (Salvador, Brazil, August 15 - 19, 2005). SIGIR '05. ACM, New York, NY, 154-161.
- [80] Joint Task Force on Computing Curricula. *Computing Curricula 2001: Computer Science Final Report*. IEEE Computer Society and the Association for Computing Machinery, Dec. 2001.
- [81] *Journal of Blacks in Higher Education*. 2006. http://www.jbhe.com/news_views (accessed August 16, 2011).
- [82] "Karel J. Robot IDE - Documentation." <http://www.st.informatik.tu-darmstadt.de/pages/lectures/inf1/kareljide/KarelJIJEDocumentation.html> (accessed December 8, 2009).
- [83] "KEDIT Text Editor for Windows." *ACADEMIC COMPUTING and COMMUNICATIONS CENTER*. September 29, 2000. <http://www.uic.edu/depts/acc/software/the/quickk.html> (accessed December 15, 2009).

- [84] Kelleher, C., Pausch, R. and Kiesler, S. 2007. Storytelling Alice motivates middle school girls to learn computer programming. In *Proceedings of the SIGCHI conference on Human factors in computing systems (CHI '07)*. ACM, New York, NY, USA, 1455-1464.
- [85] Kelleher, C. and Pausch, R. 2005. Lowering the barriers to programming: A taxonomy of programming environments and languages for novice programmers. *ACM Comput. Surv.* 37, 2 (Jun. 2005), 83-137.
- [86] Kelly, T. and Buckley, J. 2006. A Context-Aware Analysis Scheme for Bloom's Taxonomy. In *Proceedings of the 14th IEEE international Conference on Program Comprehension (June 14 - 16, 2006)*. ICPC. IEEE Computer Society, Washington, DC, 275-284.
- [87] Knudsen, Jonathan. *The Unofficial Guide to LEGO MINDSTORMS Robots*. O'Reilly Media, 1999.
- [88] Kölling, M., Bruce, Q., Andrew, P., & Rosenberg, J. The BlueJ system and its pedagogy. *Journal of Computer Science Education*, 13(4), 2003, 249--268.
- [89] Lawhead, P. B., Duncan, M. E., Bland, C. G., Goldweber, M., Schep, M., Barnes, D. J., and Hollingsworth, R. G. 2002. A road map for teaching introductory programming using LEGO© mindstorms robots. In *Working Group Reports From ITiCSE on innovation and Technology in Computer Science Education (Aarhus, Denmark, June 24 - 28, 2002)*. ITiCSE-WGR '02. ACM, New York, NY, 191-201.
- [90] Lecky-Thompson, G. "Console/Command Line Programming." *Suite101.com*. April 7, 2006. <http://computerprogramming.suite101.com/article.cfm/consoleprogramming>.
- [91] *LEGO Mindstorms*. 2009. <http://mindstorms.lego.com/en-us/Default.aspx> (accessed November 10, 2009).
- [92] Letovsky, S. 1986. Cognitive processes in program comprehension. In *Papers Presented At the First Workshop on Empirical Studies of Programmers on Empirical Studies of Programmers* (Washington, D.C., United States). E. Soloway and S. Iyengar, Eds. Ablex Publishing Corp., Norwood, NJ, 58-79
- [93] Levy, Ronit B, and Mordechai Ben-Ari. "A Survey of Research on the Jeliot Program Animation System." *Proceedings of the Chais conference on instructional technologies research 2009*. Raanana: The Open University of Israel, 2009. 41-47.
- [94] Lewis, C. 2010. How programming environment shapes perception, learning and goals: logo vs. scratch. In *Proceedings of the 41st ACM technical symposium on Computer science education (SIGCSE '10)*. ACM, New York, NY, USA, 346-350.

- [95] Lorenzen, T. and Sattar, A. 2008. Objects first using Alice to introduce object constructs in CS1. *SIGCSE Bull.* 40, 2 (Jun. 2008), 62-64.
- [96] Lumsden, L.S.: Student motivation to learn (ERIC Digest No. 92). Eugene, OR: ERIC Clearinghouse on Educational Management. (ERIC Document Reproduction Service No. ED 370200). 1994.
- [97] Maloney, J., Peppler, K., Kafai, Y., Resnick, M., and Rusk, N. 2008. Programming by choice: urban youth learning programming with scratch. In *Proceedings of the 39th SIGCSE technical symposium on Computer science education* (SIGCSE '08). ACM, New York, NY, USA, 367-371.
- [98] McMeekin, D. A., Kinsky, B. R., Chang, E., and Cooper, D. J. 2008. Checklist Inspections and Modifications: Applying Bloom's Taxonomy to Categorize Developer Comprehension. In *Proceedings of the 2008 the 16th IEEE international Conference on Program Comprehension* (June 10 - 13, 2008). ICPC. IEEE Computer Society, Washington, DC, 224-229.
- [99] McWhorter, W. I. and O'Connor, B. C. 2009. Do LEGO® Mindstorms® motivate students in CS1?. In *Proceedings of the 40th ACM Technical Symposium on Computer Science Education* (Chattanooga, TN, USA, March 04 - 07, 2009). SIGCSE '09. ACM, New York, NY, 438-442.
- [100] Merchant, J., Morrissette, R., and Porterfield, J. L. Remote measurement of eye direction allowing subject motion over one cubic foot of space. *IEEE Trans. Biomed. Eng.* BME-21, 4 (Jul. 1974), 309-317.
- [101] Microsoft. *Microsoft Visual Studio: An Overview of Microsoft® Visual Studio® 2008 White Paper*. Overview, Microsoft Corporation, 2007.
- [102] *Microsoft Visual Studio*. 2009. <http://msdn.microsoft.com/en-us/vstudio/default.aspx> (accessed December 15, 2009).
- [103] Moolenaar, Bram. *Vim: The Editor*. <http://www.vim.org/about.php> (accessed December 15, 2009).
- [104] Moreno, A., Myller, N., and Bednarik, R. 2005. Jeliot3, an extensible tool for program visualization, *5th Annual Finnish / Baltic Sea Conference on Computer Science Education*.
- [105] Moskal, B., Lurie, D. and Cooper, S. 2004. Evaluating the effectiveness of a new instructional approach. In *Proceedings of the 35th SIGCSE Technical Symposium on Computer Science Education* (Norfolk, Virginia, USA, March 03 - 07, 2004). SIGCSE '04. ACM, New York, NY, 75-79.

- [106] *Multi-Edit Version for Windows*. 2004. <http://www.hallogram.com/multedit/> (accessed December 15, 2009).
- [107] Murphy, C.A., Coover, D & Owen, S.V. (1989), Development and validation of the computer selfefficacy scale, *Educational and Psychological Measurement* 49, pp.893-899.
- [108] Nanja, M. and Cook, C. R. 1987. An analysis of the on-line debugging process. In *Empirical Studies of Programmers: Second Workshop*, G. M. Olson, S. Sheppard, and E. Soloway, Eds. Ablex Series Of Monographs, Edited Volumes, And Texts. Ablex Publishing Corp., Norwood, NJ, 172-184.
- [109] Nelson, M. 2001. Robocode, IBM Advanced Technologies. Available at HYPERLINK "<http://robocode.alphaworks.ibm.com/home/home.html>"
- [110] *NetBeans*. 2009. <http://netbeans.org/> (accessed November 10, 2009).
- [111] Nielsen, Jakob. *Be Succinct! (Writing for the Web)*. March 15, 1997. <http://www.useit.com/> (accessed August 15, 2009).
- [112] Norris, C., Barry, F., Fenwick Jr., J. B., Reid, K., and Rountree, J. 2008. ClockIt: collecting quantitative data on how beginning software developers really work. In *Proceedings of the 13th Annual Conference on innovation and Technology in Computer Science Education* (Madrid, Spain, June 30 - July 02, 2008). ITiCSE '08. ACM, New York, NY, 37-41
- [113] O'Brien, M. P., Buckley, J., and Shaft, T. M. 2004. Expectation-based, inference-based, and bottom-up software comprehension: Research Articles. *J. Softw. Maint. Evol.* 16, 6 (Nov. 2004), 427-447.
- [114] Olan, M. 2004. Dr. J vs. the bird: Java IDE's one-on-one. *J. Comput. Small Coll.* 19, 5 (May. 2004), 44-52.
- [115] *Oracle JDeveloper*. 2009. <http://www.oracle.com/technology/products/jdev/index.html> (accessed November 15, 2009).
- [116] Papert, S. 1980 *Mindstorms: Children, Computers, and Powerful Ideas*. Basic Books, Inc.
- [117] Parrish, A., Cordes, D., Lester, C., and Moore, D. 1996. Active learning and process assessment: two experiments in an Ada-based software engineering course. In *Proceedings of the Conference on Tri-Ada '96: Disciplined Software Development with Ada* (Philadelphia, Pennsylvania, United States, December 03 - 07, 1996). S. Carlson, Ed. TRI-Ada '96. ACM, New York, NY, 157-161.

- [118] Pattis, Richard E. *Karel The Robot: A Gentle Introduction to the Art of Programming, 2nd Edition*. Wiley, 1995.
- [119] Pears, A., Seidman, S., Malmi, L., Mannila, L., Adams, E., Bennedsen, J., Devlin, M., and Paterson, J. 2007. A survey of literature on the teaching of introductory programming. *SIGCSE Bull.* 39, 4 (Dec. 2007), 204-223.
- [120] Pennington, N. Stimulus structures and mental representations in expert comprehension of computer programs. *Cognitive Psychology* 19, (1987), 295-341.
- [121] Petre, M. and Blackwell, A. F. 2007. Children as Unwitting End-User Programmers. In *Proceedings of the IEEE Symposium on Visual Languages and Human-Centric Computing* (September 23 - 27, 2007). VLHCC. IEEE Computer Society, Washington, DC, 239-242.
- [122] "Pico Editor Quick Reference."
<http://www.uni.edu/its/us/document/sun/picoqref.html#qref> (accessed December 16, 2009).
- [123] Pintrich P., Smith D., Garcia T., McKeachie W.: A Manual for the Use of the Motivated Strategies for Learning Questionnaire. Technical Report 91-B-004. The Regents of the University of Michigan. 1991.
- [124] Piotrowski, J. A. 1989. Abstract machines in Miranda. *SIGCSE Bull.* 21, 3 (Sep. 1989), 44-47.
- [125] *PLT Scheme*. <http://www.plt-scheme.org/> (accessed October 15, 2009).
- [126] Pyscripter –Python IDE. 2010. <http://www.findbestopensource.com/product/pyscripter> (accessed: May 5, 2011).
- [127] Radenski, A. 2006. "Python first": a lab-based digital introduction to computer science. In *Proceedings of the 11th Annual SIGCSE Conference on innovation and Technology in Computer Science Education* (Bologna, Italy, June 26 - 28, 2006). ITICSE '06.
- [128] Rankin, Bob. "LINUX TEXT EDITORS: Can Pico Slow the Rotation of the Earth?" 2009. <http://lowfatlinux.com/linux-editor-pico.html> (accessed December 15, 2009).
- [129] Ramalingam, V., LaBelle, D., and Wiedenbeck, S. 2004. Self-efficacy and mental models in learning to program. *SIGCSE Bull.* 36, 3 (Sep. 2004), 171-175.
- [130] Ramalingam, V., LaBelle, D., and Wiedenbeck, S. 2004. Self-efficacy and mental models in learning to program. In *Proceedings of the 9th Annual SIGCSE Conference on innovation and Technology in Computer Science Education* (Leeds, United Kingdom, June 28 - 30, 2004). ITiCSE '04. ACM, New York, NY, 171-175.

- [131] Ramalingam, V. and Wiedenbeck, S. 1997. An empirical study of novice program comprehension in the imperative and object-oriented styles. In *Papers Presented At the Seventh Workshop on Empirical Studies of Programmers* (Alexandria, Virginia, United States). S. Wiedenbeck and J. Scholtz, Eds. ESP '97. ACM, New York, NY, 124-139.
- [132] Ranum, D., Miller, B., Zelle, J., and Guzdial, M. 2006. Successful approaches to teaching introductory computer science courses with python. In *Proceedings of the 37th SIGCSE Technical Symposium on Computer Science Education* (Houston, Texas, USA, March 03 - 05, 2006). SIGCSE '06. ACM, New York, NY, 396-397.
- [133] Reges, S. 2006. Back to basics in CS1 and CS2. *SIGCSE Bull.* 38, 1 (Mar. 2006), 293-297.
- [134] Reid, R. J. 1993. The object oriented paradigm in CS 1. *SIGCSE Bull.* 25, 1 (Mar. 1993), 265-269.
- [135] Reis, C. and Cartwright, R. 2004. Taming a professional IDE for the classroom. In *Proceedings of the 35th SIGCSE Technical Symposium on Computer Science Education* (Norfolk, Virginia, USA, March 03 - 07, 2004). SIGCSE '04. ACM, New York, NY, 156-160.
- [136] Rigby, P. C. and Thompson, S. 2005. Study of novice programmers using Eclipse and Gild. In *Proceedings of the 2005 OOPSLA Workshop on Eclipse Technology Exchange* (San Diego, California, October 16 - 17, 2005). eclipse '05. ACM, New York, NY, 105-109.
- [137] Ritchie, D. M. 1996. The development of the C programming language. In *History of Programming languages---II*, T. J. Bergin and R. G. Gibson, Eds. ACM, New York, NY, 671-698.
- [138] Roberts, E. 2001. An overview of MiniJava. *SIGCSE Bull.* 33, 1 (Mar. 2001), 1-5.
- [139] Romero, P., Cox, R., Boulay, B. d., and Lutz, R. 2002. Visual Attention and Representation Switching During Java Program Debugging: A Study Using the Restricted Focus Viewer. In *Proceedings of the Second international Conference on Diagrammatic Representation and inference* (April 18 - 20, 2002). M. Hegarty, B. Meyer, and N. H. Narayanan, Eds. Lecture Notes In Computer Science, vol. 2317. Springer-Verlag, London, 221-235.
- [140] Rosenberg, M. 1965. Society and the adolescent self-image. Princeton, NJ: Princeton University Press.
- [141] Rossum, R. 1999. IDLE An Integrated Development Environment in and for Python. <http://www.python.org/doc/essays/ppt/os99idle/index.htm> (accessed May 5, 2011).

- [142] Sanders, D. and Dorn, B. 2003. Jeroo: a tool for introducing object-oriented programming. In *Proceedings of the 34th SIGCSE Technical Symposium on Computer Science Education* (Reno, Nevada, USA, February 19 - 23, 2003). SIGCSE '03. ACM, New York, NY, 201-204.
- [143] Schneider, G. M. 1978. The introductory programming course in computer science: ten principles. In *Papers of the SIGCSE/CSA Technical Symposium on Computer Science Education* (Detroit, Michigan, February 23 - 24, 1978). ACM, New York, NY, 107-114.
- [144] Schulte, C. and Bennedsen, J. 2006. What do teachers teach in introductory programming?. In *Proceedings of the Second international Workshop on Computing Education Research* (Canterbury, United Kingdom, September 09 - 10, 2006). ICER '06. ACM, New York, NY, 17-28.
- [145] Seffah, Ahmed, and Eduard Metzker. *Adoption-centric Usability Engineering*. Springer, 2008.
- [146] Shaffer, D. 1986. The use of Logo in an introductory computer science course. *SIGCSE Bull.* 18, 4 (Dec. 1986), 28-31.
- [147] Shaft, T. M. 1995. Helping programmers understand computer programs: the use of metacognition. *SIGMIS Database* 26, 4 (Nov. 1995), 25-46.
- [148] Shannon, C. 2003. Another breadth-first approach to CS I using Python. In *Proceedings of the 34th SIGCSE Technical Symposium on Computer Science Education* (Reno, Nevada, USA, February 19 - 23, 2003). SIGCSE '03. ACM, New York, NY, 248-251.
- [149] Sharp, H., Y. Rogers, and J. Preece. *Interaction Design: Beyond Human-Computer Interaction*. Hoboken, NJ: John Wiley & Sons Inc., 2007.
- [150] Shneiderman, B. *Designing the user interface: Strategies for effective human-computer interaction*. Reading, Mass, Addison Wesley Longman. 1998.
- [151] Shneiderman, B. and Mayer, R. Syntactic/semantic interactions in programmer behavior: A model and experimental results. *International J. of Computer and Information Sciences* 7, 1979, 219- 239.
- [152] "SlickEdit 2009 14." *CNET: download.com*. 2009. http://download.cnet.com/SlickEdit-2009/3000-2352_4-10454612.html (accessed December 15, 2009).
- [153] Soloway, E. and Ehrlich, K. 1989. Empirical studies of programming knowledge. In *Software Reusability: Vol. 2, Applications and Experience*, T. J. Biggerstaff and A. J. Perlis, Eds. ACM, New York, NY, 235-267.

- [154] Swartz, Fred. "IDEs." *Java Notes*. 2007. <http://leepoint.net/notes-java/tools/10ide.html> (accessed February 12, 2009).
- [155] Temte, M. C. 1991. Let's begin introducing the object-oriented paradigm. *SIGCSE Bull.* 23, 1 (Mar. 1991), 73-77.
- [156] Thompson, Blair. "X2 Programmer's Editor." *Tangbu.com*. May 1, 1997. <http://www.tangbu.com/x2main.shtml> (accessed December 15, 2009).
- [157] Thompson, Blair. "X2 Editor Screenshot." *Tangbu.com*. May 1, 1997. <http://www.tangbu.com/x2main.shtml> (accessed December 15, 2009).
- [158] Torkzadeh, G. and Koufteros, X. Factorial validity of a computer self-efficacy scale and the impact of computer training. *Educational and Psychological Measurement* 54, 3 (1994), 813--821.
- [159] "Using the (The Hessling Editor) at UIC ." *ACADEMIC COMPUTING and COMMUNICATIONS CENTER* . August 17, 2005. <http://uic.edu/depts/accc/software/the/theintro.html> (accessed December 15, 2009).
- [160] van Tonder, M., Naude, K., and Cilliers, C. 2008. Jenuity: a lightweight development environment for intermediate level programming courses. In *Proceedings of the 13th Annual Conference on innovation and Technology in Computer Science Education* (Madrid, Spain, June 30 - July 02, 2008). ITiCSE '08. ACM, New York, NY, 58-62.
- [161] von Mayrhauser, A. and Vans, A. M. 1997. Program understanding behavior during debugging of large scale software. In *Papers Presented At the Seventh Workshop on Empirical Studies of Programmers* (Alexandria, Virginia, United States). S. Wiedenbeck and J. Scholtz, Eds. ESP '97. ACM, New York, NY, 157-179.
- [162] Ward, William. "Getting Started with Vi." *Linux J.*, 2003: 5.
- [163] Wiedenbeck, S., Ramalingam, V., Sarasamma, S., and Corritore, C. L. 1999. A comparison of the comprehension of object-oriented and procedural programs by novice programmers. *Interacting with Computers* 11: 255-282.
- [164] Wiedenbeck, S., Fix, V., and Scholtz, J. 1993. Characteristics of the mental representations of novice and expert programmers: an empirical study. *Int. J. Man-Mach. Stud.* 39, 5 (Nov. 1993), 793-812.
- [165] Wilson, B. C. and Shrock, S. 2001. Contributing to success in an introductory computer science course: a study of twelve factors. *SIGCSE Bull.* 33, 1 (Mar. 2001), 184-188.
- [166] Winslow, L. E. 1996. Programming pedagogy—a psychological overview. *SIGCSE Bull.* 28, 3 (Sep. 1996), 17-22.

- [167] Young, L. and Sheena, D. ‘Survey of Eye Movement Recording Methods,’ Behavior Research Methods and Instrumentation 7 pp. 397-429 (1975).
- [168] Yusuf, S., Kagdi, H., and Maletic, J. I. 2007. Assessing the Comprehension of UML Class Diagrams via Eye Tracking. In *Proceedings of the 15th IEEE international Conference on Program Comprehension* (June 26 - 29, 2007). ICPC. IEEE Computer Society, Washington, DC, 113-122.

APPENDIX A: LITERATURE REVIEW – PRIOR STUDIES

Tables 68a and 68b provide detailed summaries of prior studies related to this research.

Table 68a: Individual Evaluation

Environment	BlueJ	Alice
Reference	Hagan and Markham (2000)	Moskal, Lurie, and Cooper (2004)
Course Level	CS1	CS1
Duration of Study	1 semester	2 years
Procedure(s) Used to Obtain Data	Surveys Interviews	Grade Assessment Interviews Focus Group Discussions Surveys
Focus of Evaluation	The impact of BlueJ to teach OO concepts.	The impact of Alice on novices in an introductory programming course.
Measurements used for Evaluation	Installation Ease of Use Learning to Program Attitudes/Feelings	Grades Retention Rate Attitudes

Table 68b: Comparison Evaluation

Environment	CS1 Sandbox	Eclipse	Raptor	LEGO® Mindstorms
Reference	DePasquale (2005)	Chen and Marx (2005)	Carlisle, Wilson, Humphries, and Hadfield (2005)	McWhorter and O'Connor (2009)
Other environment(s) involved in study	Microsoft Visual C++ .Net	Ready to Program	MATLAB	<i>Authors did not provide this information</i>
Course Level	CS1	CS2	CS1	CS1
Duration of study	1 semester	2 years	3 semesters	2 semesters
Procedure(s) used to obtain data	Grade Assessment Survey Focus Group Discussion	Questionnaire	Final Exam (implement three programs) Survey	Questionnaire
Focus of Evaluation	The impact of CS1 Sandbox (with or without language subsets) on novices.	The impact of Eclipse in a CS2 course.	The effect of Raptor on novices when learning algorithmic problem solving.	The motivation of LEGO® Mindstorms on novices learning to program.
Measurements used for evaluation	Grades Number of Compilations Error Rate Time-on-task	Personal Feelings Usability Programming Experience	Correct Implementation Ease of Use (individual evaluation of Raptor)	Intrinsic and Extrinsic Motivation Task Value Control Learning Belief Self-Efficacy Test Anxiety

APPENDIX B: LITERATURE REVIEW – MEASURES USED IN PRIOR STUDIES

Table 69 displays a list of related studies and sources that have studied or employed measures of engagement, comprehension, efficiency, and usability.

Table 69: Related Studies & Sources that Employed Measures of Engagement, Comprehension, Efficiency, and/or Usability

Measures	Engagement	Comprehension	Efficiency(Time)	Usability
<p>Other studies that discussed or evaluated these measures</p>	<p>Bergin and Reilly [14] Chen and Marx [26] Hagan and Markham [62] Lumsden [96] Moskal et al. [105] McWhorter and O’Connor [99] Wilson and Shrock [166]</p>	<p>DePasquale [41] Fenwick et al. [52] Jadud [71, 72, 73] Moskal et al. [105] Norris et al. [112] Parrish et al. [117] Ramalingam and Wiedenbeck [129, 130] Wiedenbeck et al. [163]</p>	<p>Card, Thomas, and Newell [24] DePasquale [41] Fenwick et al. [52] Jadud [71, 72, 73] Norris et al. [112] Parrish et al. [117]</p>	<p>From Seffah et al.’s paper [145]: Constantine and Lockwood Nielsen Preece Shackel Shneiderman</p>
<p>Potential approaches for environment evaluation</p>	<p>Rosenberg Self-Esteem Questionnaire [140] Computer Programming Self-Efficacy Scale [130] Motivated Strategies for Learning Questionnaire (MSLQ) [123]</p>	<p>Bloom’s Taxonomy [18] Mental Models (ex. Good, Pennington, Wiedenbeck, etc.) [31, 34, 39, 55, 58, 67, 93, 113, 120, 131, 147, 151, 153, 161] Eye-tracking [10, 11, 12, 35, 37, 47, 48, 70, 74, 80, 111, 121, 139, 167, 168]</p>	<p>Keystroke-Level Model [24] “Time on task” [41, 53, 71, 72, 73, 112, 117]</p>	<p>Measures [145]: Constantine and Lockwood, Nielsen, Preece, Shackel, Shneiderman</p>

APPENDIX C: ALICEVILLE OUTREACH SURVEYS

This section of the appendix displays surveys for the Aliceville High School Outreach. This survey was used to evaluate the students' behavior with PREOP through engagement, comprehension, and usability.

PREOP EFFECIVENESS QUESTIONNAIRE

Usability & Demographics

1. How comfortable were you with the idea of programming robots?
() 1 = "not comfortable at all" () 2 = "slightly comfortable" () 3 = "50/50" () 4 = "mostly comfortable" () 5 = "absolutely comfortable"
2. Why? What factors led to your response for Question 1?
3. How easy was PREOP to use?
() 1 = "not easy at all" () 2 = "slightly easy" () 3 = "50/50" () 4 = "mostly easy" () 5 = "absolutely easy"
4. After completing today's session, how comfortable are you with programming robots?
() 1 = "not comfortable at all" () 2 = "slightly comfortable" () 3 = "50/50" () 4 = "mostly comfortable" () 5 = "absolutely comfortable"
5. Would you use PREOP in the future?
() 1 = "not at all" () 2 = "slightly yes" () 3 = "50/50" () 4 = "most likely yes" () 5 = "absolutely yes"
6. Would you use PREOP as a hobby or extracurricular activity?
() 1 = "not at all" () 2 = "slightly yes" () 3 = "50/50" () 4 = "most likely yes" () 5 = "absolutely yes"
7. Have you ever used another programming language or tool? Yes / No
 - a. If so, what was that language?
 - b. Are you currently using this language?
 - c. How much do you like using this language (scale 1 - 5)?
() 1 = "not at all" () 2 = "slightly like it" () 3 = "50/50" () 4 = "most did like" () 5 = "absolutely like it"
8. Are you interested in working in computer –related career?
() 1 = "not at all" () 2 = "slightly yes" () 3 = "50/50" () 4 = "most likely yes" () 5 = "absolutely yes"
9. Why or why not are you interested in working in computer science in the future?
10. What is your official class status?
() Senior () Junior () Sophomore () Freshman () 8th grade () Other
11. Gender
() Male () Female
12. Ethnic Background
() African American () Asian () Caucasian () Hispanic () Native American () Other (please specify) _____
13. What grades do you receive the most?
() As and Bs () Bs and Cs () Cs and Ds () Ds and Fs
14. How reliable was PREOP and the robots?
() No Problems () A few problems () Many Problems () Did not work well
15. Were frustrated by the programming process?

Comprehension – Pennington’s Model

Questions 16-20

world.my first method ()
length = 0.1 , speed = 0.5 , durationForMovement = 2
length set value to 0.5
robot move forward length ((length / speed))
robot turn right 0.5 revolutions
durationForMovement set value to ((length * 2))
robot move backward length duration = durationForMovement seconds

16. Is length initialized to (0.1)?
17. Is durationForMovement calculated before the robot moves (backward)?
18. Will the value of speed affect the amount the robot moves forward?
19. Does speed have a value before durationForMovement?
20. Does the program move the robot in a triangle?

Alternate Questions 16-20

world.my first method ()
length = 0.1 , speed = 0.5 , durationForMovement = 2
length set value to 0.5
robot move forward length duration = durationForMovement seconds
robot turn right 0.5 revolutions
durationForMovement set value to ((length * 2))
robot move backward length duration = durationForMovement seconds

16. Is length initialized to (0.2)?
17. Is durationForMovement calculated before the robot moves (backward)?
18. Will the value of speed affect the amount the robot moves forward?
19. Does speed have a value before durationForMovement?
20. Does the program move the robot in a circle?

Engagement – Self-Efficacy

21. I can create correct statements in PREOP.

1 = "not at all" 2 = "slightly yes" 3 = "50/50" 4 = "most likely yes" 5 = "absolutely yes"

22. I can write a small program given a small problem that is familiar to me in PREOP.

1 = "not at all" 2 = "slightly yes" 3 = "50/50" 4 = "most likely yes" 5 = "absolutely yes"

23. I can create a program that someone else could comprehend and add features to at a later date when using PREOP.

1 = "not at all" 2 = "slightly yes" 3 = "50/50" 4 = "most likely yes" 5 = "absolutely yes"

24. I can complete a programming project if I had enough time to complete the program.

1 = "not at all" 2 = "slightly yes" 3 = "50/50" 4 = "most likely yes" 5 = "absolutely yes"

25. I can complete a programming project once someone else helped me get started.

1 = "not at all" 2 = "slightly yes" 3 = "50/50" 4 = "most likely yes" 5 = "absolutely yes"

26. I can complete a programming project if I could ask someone for help if I got stuck.

1 = "not at all" 2 = "slightly yes" 3 = "50/50" 4 = "most likely yes" 5 = "absolutely yes"

27. I can complete a programming project if I had only the built-in help facility for assistance.

1 = "not at all" 2 = "slightly yes" 3 = "50/50" 4 = "most likely yes" 5 = "absolutely yes"

28. I can complete a programming project if I had only the PREOP reference manual for help.

1 = "not at all" 2 = "slightly yes" 3 = "50/50" 4 = "most likely yes" 5 = "absolutely yes"

29. I can complete a programming project if someone showed me how to solve the problem first.

1 = "not at all" 2 = "slightly yes" 3 = "50/50" 4 = "most likely yes" 5 = "absolutely yes"

30. I can find ways of overcoming the problem if I got stuck at a point while working on a program in PREOP.

1 = "not at all" 2 = "slightly yes" 3 = "50/50" 4 = "most likely yes" 5 = "absolutely yes"

31. I can find a way to concentrate on a problem, even when there are many distractions around me.

1 = "not at all" 2 = "slightly yes" 3 = "50/50" 4 = "most likely yes" 5 = "absolutely yes"

32. I can find ways of motivating myself to solve problems, even if the problem area is of no interest of me.

1 = "not at all" 2 = "slightly yes" 3 = "50/50" 4 = "most likely yes" 5 = "absolutely yes"

33. I can come up with a suitable strategy for a given programming project in PREOP in a short time.
() 1 = "not at all" () 2 = "slightly yes" () 3 = "50/50" () 4 = "most likely yes" () 5 = "absolutely yes"
34. There is usually one correct approach to a problem
() 1 = "Strongly disagree" () 2 = "Somewhat disagree" () 3 = "50/50" () 4 = "Mostly agree" () 5 = "Absolutely agree"
35. I am not satisfied until I understand how something works
() 1 = "Strongly disagree" () 2 = "Somewhat disagree" () 3 = "50/50" () 4 = "Mostly agree" () 5 = "Absolutely agree"
36. Nearly everyone is capable of understanding computers if they work at it
() 1 = "Strongly disagree" () 2 = "Somewhat disagree" () 3 = "50/50" () 4 = "Mostly agree" () 5 = "Absolutely agree"
37. I do not spend more than five minutes on a problem before I ask someone for help
() 1 = "Strongly disagree" () 2 = "Somewhat disagree" () 3 = "50/50" () 4 = "Mostly agree" () 5 = "Absolutely agree"
38. A significant problem in learning is memorizing all the information I need to know
() 1 = "Strongly disagree" () 2 = "Somewhat disagree" () 3 = "50/50" () 4 = "Mostly agree" () 5 = "Absolutely agree"
39. What we learn in school has little relation to the real world
() 1 = "Strongly disagree" () 2 = "Somewhat disagree" () 3 = "50/50" () 4 = "Mostly agree" () 5 = "Absolutely agree"
40. I enjoy solving problems
() 1 = "Strongly disagree" () 2 = "Somewhat disagree" () 3 = "50/50" () 4 = "Mostly agree" () 5 = "Absolutely agree"
41. It is possible to solve a problem in two different ways and get two different results
() 1 = "Strongly disagree" () 2 = "Somewhat disagree" () 3 = "50/50" () 4 = "Mostly agree" () 5 = "Absolutely agree"

APPENDIX D: CS1-LABORATORY SURVEYS

This section of the appendix displays the surveys for the CS1-Laboratory Study. These surveys were used to evaluate the students' behavior with IDLE, PyScripter, and Notepad through engagement, comprehension, efficiency and usability.

Self –Efficacy /Demographics Survey

Expectations

1. Write a syntactically correct Python program

- 1 = "not confident at all" 4 = "50/50" 5 = "fairly confident"
 2 = "mostly not confident" 6 = "mostly confident"
 3 = "slightly confident" 7 = "absolutely confident"

2. Understand the language structure of a Python program and the usage of reserved words

- 1 = "not confident at all" 4 = "50/50" 5 = "fairly confident"
 2 = "mostly not confident" 6 = "mostly confident"
 3 = "slightly confident" 7 = "absolutely confident"

3. Write syntactically correct blocks of code using Python

- 1 = "not confident at all" 4 = "50/50" 5 = "fairly confident"
 2 = "mostly not confident" 6 = "mostly confident"
 3 = "slightly confident" 7 = "absolutely confident"

4. Write a Python program that displays a greeting message

- 1 = "not confident at all" 4 = "50/50" 5 = "fairly confident"
 2 = "mostly not confident" 6 = "mostly confident"
 3 = "slightly confident" 7 = "absolutely confident"

5. Write a Python program that computers the average of three numbers

- 1 = "not confident at all" 4 = "50/50" 5 = "fairly confident"
 2 = "mostly not confident" 6 = "mostly confident"
 3 = "slightly confident" 7 = "absolutely confident"

6. Use built-in functions that are available in the various Python libraries

- 1 = "not confident at all" 4 = "50/50" 5 = "fairly confident"
 2 = "mostly not confident" 6 = "mostly confident"
 3 = "slightly confident" 7 = "absolutely confident"

7. Build my own Python library

- 1 = "not confident at all" 4 = "50/50" 5 = "fairly confident"
 2 = "mostly not confident" 6 = "mostly confident"
 3 = "slightly confident" 7 = "absolutely confident"

8. Write a small Python program given a small problem that is familiar to me

- 1 = "not confident at all" 4 = "50/50" 5 = "fairly confident"
 2 = "mostly not confident" 6 = "mostly confident"
 3 = "slightly confident" 7 = "absolutely confident"

9. Write a reasonably sized Python program that can solve a problem that is only vaguely familiar to me

- 1 = "not confident at all" 4 = "50/50" 5 = "fairly confident"
 2 = "mostly not confident" 6 = "mostly confident"
 3 = "slightly confident" 7 = "absolutely confident"

10. Write a long and complex Python program to solve any given problem as long as the specifications are clearly defined

- 1 = "not confident at all" 4 = "50/50" 5 = "fairly confident"
 2 = "mostly not confident" 6 = "mostly confident"
 3 = "slightly confident" 7 = "absolutely confident"

11. Organize and design my own program in a logical manner

- 1 = "not confident at all" 4 = "50/50" 5 = "fairly confident"
 2 = "mostly not confident" 6 = "mostly confident"
 3 = "slightly confident" 7 = "absolutely confident"

12. Understand object-oriented paradigm

- 1 = "not confident at all" 4 = "50/50" 5 = "fairly confident"
 2 = "mostly not confident" 6 = "mostly confident"
 3 = "slightly confident" 7 = "absolutely confident"

13. Identify the objects in the problem domain and declare, define and use them

- 1 = "not confident at all" 4 = "50/50" 5 = "fairly confident"
 2 = "mostly not confident" 6 = "mostly confident"
 3 = "slightly confident" 7 = "absolutely confident"

14. Make use of a pre-written function, given a clearly labeled declaration of the function

- 1 = "not confident at all" 4 = "50/50" 5 = "fairly confident"
 2 = "mostly not confident" 6 = "mostly confident"
 3 = "slightly confident" 7 = "absolutely confident"

15. Make use of a class that is already defined, given a clearly labeled declaration of the class

- 1 = "not confident at all" 4 = "50/50" 5 = "fairly confident"
 2 = "mostly not confident" 6 = "mostly confident"
 3 = "slightly confident" 7 = "absolutely confident"

16. Debug (correct all the errors) a long and complex program that I had written and make it work

- 1 = "not confident at all" 4 = "50/50" 5 = "fairly confident"
 2 = "mostly not confident" 6 = "mostly confident"
 3 = "slightly confident" 7 = "absolutely confident"

17. Comprehend a long, complex multi-file program

- 1 = "not confident at all" 4 = "50/50" 5 = "fairly confident"
 2 = "mostly not confident" 6 = "mostly confident"
 3 = "slightly confident" 7 = "absolutely confident"

18. Complete a programming project if someone showed me how to solve the problem first

- 1 = "not confident at all" 4 = "50/50" 5 = "fairly confident"
 2 = "mostly not confident" 6 = "mostly confident"
 3 = "slightly confident" 7 = "absolutely confident"

19. Complete a programming project if I had only the language reference manual for help

- 1 = "not confident at all" 4 = "50/50" 5 = "fairly confident"
 2 = "mostly not confident" 6 = "mostly confident"
 3 = "slightly confident" 7 = "absolutely confident"

20. Complete a programming project if I could call someone for help if I got stuck

- 1 = "not confident at all" 4 = "50/50" 5 = "fairly confident"
 2 = "mostly not confident" 6 = "mostly confident"
 3 = "slightly confident" 7 = "absolutely confident"

21. Complete a programming project once someone showed me how to get it started

- 1 = "not confident at all" 4 = "50/50" 5 = "fairly confident"
 2 = "mostly not confident" 6 = "mostly confident"
 3 = "slightly confident" 7 = "absolutely confident"

22. Complete a programming project if I had a lot of time to complete the program

- 1 = "not confident at all" 4 = "50/50" 5 = "fairly confident"
 2 = "mostly not confident" 6 = "mostly confident"
 3 = "slightly confident" 7 = "absolutely confident"

23. Complete a programming project if I had just the built-in help facility for assistance

- 1 = "not confident at all" 4 = "50/50" 5 = "fairly confident"
 2 = "mostly not confident" 6 = "mostly confident"
 3 = "slightly confident" 7 = "absolutely confident"

24. Find ways of overcoming the problem if I got stuck at a point while working on a programming project

- 1 = "not confident at all" 4 = "50/50" 5 = "fairly confident"
 2 = "mostly not confident" 6 = "mostly confident"
 3 = "slightly confident" 7 = "absolutely confident"

25. Come up with a suitable strategy for a given programming project in a short time

- 1 = "not confident at all" 4 = "50/50" 5 = "fairly confident"
 2 = "mostly not confident" 6 = "mostly confident"
 3 = "slightly confident" 7 = "absolutely confident"

26. Manage my time efficiently if I had a pressing deadline on a programming project

- 1 = "not confident at all" 4 = "50/50" 5 = "fairly confident"
 2 = "mostly not confident" 6 = "mostly confident"
 3 = "slightly confident" 7 = "absolutely confident"

27. Mentally trace through the execution of a long, complex, multi-file program given to me

- 1 = "not confident at all" 4 = "50/50" 5 = "fairly confident"
 2 = "mostly not confident" 6 = "mostly confident"
 3 = "slightly confident" 7 = "absolutely confident"

28. Rewrite lengthy confusing portions of code to be more readable and clear

- 1 = "not confident at all" 4 = "50/50" 5 = "fairly confident"
 2 = "mostly not confident" 6 = "mostly confident"
 3 = "slightly confident" 7 = "absolutely confident"

29. Find a way to concentrate on my program, even when there were many distractions around me

- 1 = "not confident at all" 4 = "50/50" 5 = "fairly confident"
 2 = "mostly not confident" 6 = "mostly confident"
 3 = "slightly confident" 7 = "absolutely confident"

30. Find ways of motivating myself to program, even if the problem was of no interest to me

- 1 = "not confident at all" 4 = "50/50" 5 = "fairly confident"
 2 = "mostly not confident" 6 = "mostly confident"
 3 = "slightly confident" 7 = "absolutely confident"

31. Write a program that someone else could comprehend and add features to at a later date

- 1 = "not confident at all" 4 = "50/50" 5 = "fairly confident"
 2 = "mostly not confident" 6 = "mostly confident"
 3 = "slightly confident" 7 = "absolutely confident"

Demographics

32. How would you rate your programming skill in comparison to others in your class?

- I have a lot more skill
 I have somewhat more skill
 I have average skill
 I have somewhat less skill
 I have a lot less skill

33. How would you rate your computer knowledge in comparison to others in your class?

- I have a lot more knowledge
 I have somewhat more knowledge
 I have average knowledge
 I have somewhat less knowledge
 I have a lot less knowledge

34. Are you intimidated by programming?

- Yes
 No

35. Are you intimidated by Computer Science?

- Yes
 No

36. What semester of college are you in?

- 1st
 2nd
 3rd
 4th
 5th-6th
 7th +

37. What is your current major?

- Computer Science
 Electrical Engineering
 Computer Engineering
 Management Information Systems
 Math
 Other _____

38. What is your current GPA?

- First semester
- < 1.0
- 1.0 - 2.0
- 2.0 - 3.0
- 3.0- 4.0

39. What is your official class status?

- Senior
- Junior
- Sophomore
- Freshman
- Other

40. Are you a transfer student (i. e. did you start your freshman year somewhere other than The University of Alabama)?

- Yes
- No

41. Are you currently taking CS150?

- Yes
- No
- Dropped it
- Completed

42. Is CS150 your first programming class?

- Yes
- No - High school course
- No - Other College course
- Not taking CS150

43. What grade do you expect in CS150?

- A+, A, A-
- B+, B, B-
- C+, C, C-
- D+, D, D-
- F
- Not taking CS150

44. Do you need another programming class to graduate?

- Yes
- No

45. When do you plan to take the next programming course?

- Next semester
- Another semester
- Never

46. Gender

- Male
- Female

Assignment – Time on Task

Instructions

Write a program that converts 700 days into y years, m months, and d days remaining. Use your Python environment to complete this task.

Pennington's Model – Version 1

PLEASE PROVIDE NAME AND CWID

NAME: _____ CWID: _____

```
#Python program version 1
from math import *
def main():
    amount = eval (input("Enter an amount of change from 1 to 99 cents. "))
    quarters = amount / 25
    amountLeft = amount % 25
    dimes = amountLeft / 10
    amountLeft = amountLeft % 10
    nickels = amountLeft / 5
    amountLeft = amountLeft % 5
    pennies = amountLeft

    print ("There are ", quarters , "Quarters, ",dimes ,"Dimes, ",
           nickels ,"Nickels, and ",pennies ,"Pennies " )
main()
```

- 1: Is the variable pennies initialized to 0?[Yes / No]
2. Is the number of quarters needed calculated before the number of dimes needed?..... [Yes / No]
3. Will the value of amountLeft affect the value of pennies?[Yes / No]
4. Does amountLeft have a value before quarters is assigned a value?[Yes / No]
5. Does this program compute how to give change in the largest possible denominations? [Yes / No]

Pennington's Model – Version 2

PLEASE PROVIDE NAME AND CWID

NAME: _____ CWID: _____

```
#Python program version 2
from math import *
def main():
    amount = eval (input("Enter an amount of change from 1 to 99 cents. "))
    quarters = amount / 25
    amountLeft = amount % 25
    dimes = amountLeft / 10
    amountLeft = amountLeft % 10
    nickels = amountLeft / 5
    amountLeft = amountLeft % 5
    pennies = amountLeft

    print ("There are ", quarters , "Quarters, ",dimes , "Dimes, ",
           nickels , "Nickels, and ",pennies , "Pennies " )
main()
```

1. Is the variable Pennies initialized to 25?[Yes / No]
2. Is the number of quarters needed calculated after the number of dimes needed?[Yes / No]
3. Will the value of AmountLeft affect the value of Nickels?.....[Yes / No]
4. Does AmountLeft have a value after Quarters is assigned a value?.....[Yes / No]
5. Does this program compute how to give change in the smallest possible denominations?..[Yes / No]

Usability – IDLE

Questions about Your Environment

1.) What was your initial response to using IDLE to program?

1a.) Why? What factors led you to this?

2.) What is the easiest thing about using IDLE to program?

3.) What is the hardest thing about using IDLE to program?

4.) How comfortable are you with using IDLE for programming?

- 1 = "not comfortable at all" 4 = "50/50" 5 = "fairly comfortable"
 2 = "mostly not comfortable" 6 = "mostly comfortable"
 3 = "slightly comfortable" 7 = "absolutely comfortable"

5.) If you were asked to complete another program assignment, how confident are you in writing one using IDLE at this point?

- 1 = "not confident at all" 4 = "50/50" 5 = "fairly confident"
 2 = "mostly not confident" 6 = "mostly confident"
 3 = "slightly confident" 7 = "absolutely confident"

6.) How much do you like IDLE after using it?

- 1 = "not at all" 4 = "50/50" 5 = "fairly like"
 2 = "mostly do not like" 6 = "mostly like"
 3 = "slightly like" 7 = "absolutely like"

6a.) Why? What factors led you to this decision?

7.) Are you currently using another programming environment?

8.) What "other" programming environment(s) are you using?

9.) Is the "other" programming environment mandatory for a course you are taking?

10.) How much do you enjoy using this "other" programming environment?

- 1 = "not at all" 4 = "50/50" 5 = "fairly like"
 2 = "mostly do not like" 6 = "mostly like"
 3 = "slightly like" 7 = "absolutely like"

11.) Between IDLE and your "other" environment, which environment do you like better?

11a.) Why? What factors led you to this?

12.) Would you use your "other" programming environment for random projects outside of a course? Why or why not?

13.) Would you use IDLE for random projects outside of a course? Why or why not?

Thank You!

Thank you for your participation

Usability – PYSCRIPTER

Questions about Your Environment

1.) What was your initial response to using PYSCRIPTER to program?

1a.) Why? What factors led you to this?

2.) What is the easiest thing about using PYSCRIPTER to program?

3.) What is the hardest thing about using PYSCRIPTER to program?

4.) How comfortable are you with using PYSCRIPTER for programming?

- 1 = "not comfortable at all" 4 = "50/50" 5 = "fairly comfortable"
 2 = "mostly not comfortable" 6 = "mostly comfortable"
 3 = "slightly comfortable" 7 = "absolutely comfortable"

5.) If you were asked to complete another program assignment, how confident are you in writing one using PYSCRIPTER at this point?

- 1 = "not confident at all" 4 = "50/50" 5 = "fairly confident"
 2 = "mostly not confident" 6 = "mostly confident"
 3 = "slightly confident" 7 = "absolutely confident"

6.) How much do you like PYSCRIPTER after using it?

- 1 = "not at all" 4 = "50/50" 5 = "fairly like"
 2 = "mostly do not like" 6 = "mostly like"
 3 = "slightly like" 7 = "absolutely like"

6a.) Why? What factors led you to this decision?

7.) Are you currently using another programming environment?

8.) What "other" programming environment(s) are you using?

9.) Is the "other" programming environment mandatory for a course you are taking?

10.) How much do you enjoy using this "other" programming environment?

- 1 = "not at all" 4 = "50/50" 5 = "fairly like"
 2 = "mostly do not like" 6 = "mostly like"
 3 = "slightly like" 7 = "absolutely like"

11.) Between PYSCRIPTER and your "other" environment, which environment do you like better?

11a.) Why? What factors led you to this?

12.) Would you use your "other" programming environment for random projects outside of a course? Why or why not?

13.) Would you use PYSCRIPTER for random projects outside of a course? Why or why not?

Thank You!

Thank you for your participation

Usability – NOTEPAD/COMMAND PROMPT

Questions about Your Environment

1.) What was your initial response to using NOTEPAD/COMMAND PROMPT to program?

1a.) Why? What factors led you to this?

2.) What is the easiest thing about using NOTEPAD/COMMAND PROMPT to program?

3.) What is the hardest thing about using NOTEPAD/COMMAND PROMPT to program?

4.) How comfortable are you with using NOTEPAD/COMMAND PROMPT for programming?

- 1 = "not comfortable at all" 4 = "50/50" 5 = "fairly comfortable"
 2 = "mostly not comfortable" 6 = "mostly comfortable"
 3 = "slightly comfortable" 7 = "absolutely comfortable"

5.) If you were asked to complete another program assignment, how confident are you in writing one using NOTEPAD/COMMAND PROMPT at this point?

- 1 = "not confident at all" 4 = "50/50" 5 = "fairly confident"
 2 = "mostly not confident" 6 = "mostly confident"
 3 = "slightly confident" 7 = "absolutely confident"

6.) How much do you like NOTEPAD/COMMAND PROMPT after using it?

- 1 = "not at all" 4 = "50/50" 5 = "fairly like"
 2 = "mostly do not like" 6 = "mostly like"
 3 = "slightly like" 7 = "absolutely like"

6a.) Why? What factors led you to this decision?

7.) Are you currently using another programming environment?

8.) What "other" programming environment(s) are you using?

9.) Is the "other" programming environment mandatory for a course you are taking?

10.) How much do you enjoy using this "other" programming environment?

- 1 = "not at all" 4 = "50/50" 5 = "fairly like"
 2 = "mostly do not like" 6 = "mostly like"
 3 = "slightly like" 7 = "absolutely like"

11.) Between NOTEPAD/COMMAND PROMPT and your "other" environment, which environment do you like better?

11a.) Why? What factors led you to this?

12.) Would you use your "other" programming environment for random projects outside of a course? Why or why not?

13.) Would you use NOTEPAD/COMMAND PROMPT for random projects outside of a course? Why or why not?

Thank You!

Thank you for your participation

APPENDIX E: CS1 SURVEYS

This section of the appendix displays the surveys for the CS1 Study. These surveys were used to evaluate the students' behavior with IDLE and VIM through engagement, comprehension, efficiency, and usability.

Self –Efficacy (1st, 2nd, & 3rd Assessments)

Expectations

1. Write a syntactically correct Python program

- 1 = "not confident at all" 4 = "50/50" 5 = "fairly confident"
 2 = "mostly not confident" 6 = "mostly confident"
 3 = "slightly confident" 7 = "absolutely confident"

2. Understand the language structure of a Python program and the usage of reserved words

- 1 = "not confident at all" 4 = "50/50" 5 = "fairly confident"
 2 = "mostly not confident" 6 = "mostly confident"
 3 = "slightly confident" 7 = "absolutely confident"

3. Write syntactically correct blocks of code using Python

- 1 = "not confident at all" 4 = "50/50" 5 = "fairly confident"
 2 = "mostly not confident" 6 = "mostly confident"
 3 = "slightly confident" 7 = "absolutely confident"

4. Write a Python program that displays a greeting message

- 1 = "not confident at all" 4 = "50/50" 5 = "fairly confident"
 2 = "mostly not confident" 6 = "mostly confident"
 3 = "slightly confident" 7 = "absolutely confident"

5. Write a Python program that computers the average of three numbers

- 1 = "not confident at all" 4 = "50/50" 5 = "fairly confident"
 2 = "mostly not confident" 6 = "mostly confident"
 3 = "slightly confident" 7 = "absolutely confident"

6. Use built-in functions that are available in the various Python libraries

- 1 = "not confident at all" 4 = "50/50" 5 = "fairly confident"
 2 = "mostly not confident" 6 = "mostly confident"
 3 = "slightly confident" 7 = "absolutely confident"

7. Build my own Python library

- 1 = "not confident at all" 4 = "50/50" 5 = "fairly confident"
 2 = "mostly not confident" 6 = "mostly confident"
 3 = "slightly confident" 7 = "absolutely confident"

8. Write a small Python program given a small problem that is familiar to me

- 1 = "not confident at all" 4 = "50/50" 5 = "fairly confident"
 2 = "mostly not confident" 6 = "mostly confident"
 3 = "slightly confident" 7 = "absolutely confident"

9. Write a reasonably sized Python program that can solve a problem that is only vaguely familiar to me

- 1 = "not confident at all" 4 = "50/50" 5 = "fairly confident"
 2 = "mostly not confident" 6 = "mostly confident"
 3 = "slightly confident" 7 = "absolutely confident"

10. Write a long and complex Python program to solve any given problem as long as the specifications are clearly defined

- 1 = "not confident at all" 4 = "50/50" 5 = "fairly confident"
 2 = "mostly not confident" 6 = "mostly confident"
 3 = "slightly confident" 7 = "absolutely confident"

11. Organize and design my own program in a logical manner

- 1 = "not confident at all" 4 = "50/50" 5 = "fairly confident"
 2 = "mostly not confident" 6 = "mostly confident"
 3 = "slightly confident" 7 = "absolutely confident"

12. Understand object-oriented paradigm

- 1 = "not confident at all" 4 = "50/50" 5 = "fairly confident"
 2 = "mostly not confident" 6 = "mostly confident"
 3 = "slightly confident" 7 = "absolutely confident"

13. Identify the objects in the problem domain and declare, define and use them

- 1 = "not confident at all" 4 = "50/50" 5 = "fairly confident"
 2 = "mostly not confident" 6 = "mostly confident"
 3 = "slightly confident" 7 = "absolutely confident"

14. Make use of a pre-written function, given a clearly labeled declaration of the function

- 1 = "not confident at all" 4 = "50/50" 5 = "fairly confident"
 2 = "mostly not confident" 6 = "mostly confident"
 3 = "slightly confident" 7 = "absolutely confident"

15. Make use of a class that is already defined, given a clearly labeled declaration of the class

- 1 = "not confident at all" 4 = "50/50" 5 = "fairly confident"
 2 = "mostly not confident" 6 = "mostly confident"
 3 = "slightly confident" 7 = "absolutely confident"

16. Debug (correct all the errors) a long and complex program that I had written and make it work

- 1 = "not confident at all" 4 = "50/50" 5 = "fairly confident"
 2 = "mostly not confident" 6 = "mostly confident"
 3 = "slightly confident" 7 = "absolutely confident"

17. Comprehend a long, complex multi-file program

- 1 = "not confident at all" 4 = "50/50" 5 = "fairly confident"
 2 = "mostly not confident" 6 = "mostly confident"
 3 = "slightly confident" 7 = "absolutely confident"

18. Complete a programming project if someone showed me how to solve the problem first

- 1 = "not confident at all" 4 = "50/50" 5 = "fairly confident"
 2 = "mostly not confident" 6 = "mostly confident"
 3 = "slightly confident" 7 = "absolutely confident"

19. Complete a programming project if I had only the language reference manual for help

- 1 = "not confident at all" 4 = "50/50" 5 = "fairly confident"
 2 = "mostly not confident" 6 = "mostly confident"
 3 = "slightly confident" 7 = "absolutely confident"

20. Complete a programming project if I could call someone for help if I got stuck

- 1 = "not confident at all" 4 = "50/50" 5 = "fairly confident"
 2 = "mostly not confident" 6 = "mostly confident"
 3 = "slightly confident" 7 = "absolutely confident"

21. Complete a programming project once someone showed me how to get it started

- 1 = "not confident at all" 4 = "50/50" 5 = "fairly confident"
 2 = "mostly not confident" 6 = "mostly confident"
 3 = "slightly confident" 7 = "absolutely confident"

22. Complete a programming project if I had a lot of time to complete the program

- 1 = "not confident at all" 4 = "50/50" 5 = "fairly confident"
 2 = "mostly not confident" 6 = "mostly confident"
 3 = "slightly confident" 7 = "absolutely confident"

23. Complete a programming project if I had just the built-in help facility for assistance

- 1 = "not confident at all" 4 = "50/50" 5 = "fairly confident"
 2 = "mostly not confident" 6 = "mostly confident"
 3 = "slightly confident" 7 = "absolutely confident"

24. Find ways of overcoming the problem if I got stuck at a point while working on a programming project

- 1 = "not confident at all" 4 = "50/50" 5 = "fairly confident"
 2 = "mostly not confident" 6 = "mostly confident"
 3 = "slightly confident" 7 = "absolutely confident"

25. Come up with a suitable strategy for a given programming project in a short time

- 1 = "not confident at all" 4 = "50/50" 5 = "fairly confident"
 2 = "mostly not confident" 6 = "mostly confident"
 3 = "slightly confident" 7 = "absolutely confident"

26. Manage my time efficiently if I had a pressing deadline on a programming project

- 1 = "not confident at all" 4 = "50/50" 5 = "fairly confident"
 2 = "mostly not confident" 6 = "mostly confident"
 3 = "slightly confident" 7 = "absolutely confident"

27. Mentally trace through the execution of a long, complex, multi-file program given to me

- 1 = "not confident at all" 4 = "50/50" 5 = "fairly confident"
 2 = "mostly not confident" 6 = "mostly confident"
 3 = "slightly confident" 7 = "absolutely confident"

28. Rewrite lengthy confusing portions of code to be more readable and clear

- 1 = "not confident at all" 4 = "50/50" 5 = "fairly confident"
 2 = "mostly not confident" 6 = "mostly confident"
 3 = "slightly confident" 7 = "absolutely confident"

29. Find a way to concentrate on my program, even when there were many distractions around me

- 1 = "not confident at all" 4 = "50/50" 5 = "fairly confident"
 2 = "mostly not confident" 6 = "mostly confident"
 3 = "slightly confident" 7 = "absolutely confident"

30. Find ways of motivating myself to program, even if the problem was of no interest to me

- 1 = "not confident at all" 4 = "50/50" 5 = "fairly confident"
 2 = "mostly not confident" 6 = "mostly confident"
 3 = "slightly confident" 7 = "absolutely confident"

31. Write a program that someone else could comprehend and add features to at a later date

- 1 = "not confident at all" 4 = "50/50" 5 = "fairly confident"
 2 = "mostly not confident" 6 = "mostly confident"
 3 = "slightly confident" 7 = "absolutely confident"

Demographics (1st Assessment)

32. How would you rate your programming skill in comparison to others in your class?

- I have a lot more skill
- I have somewhat more skill
- I have average skill
- I have somewhat less skill
- I have a lot less skill

33. How would you rate your computer knowledge in comparison to others in your class?

- I have a lot more knowledge
- I have somewhat more knowledge
- I have average knowledge
- I have somewhat less knowledge
- I have a lot less knowledge

34. Are you intimidated by programming?

- Yes
- No

35. Are you intimidated by Computer Science?

- Yes
- No

36. What semester of college are you in?

- 1st
- 2nd
- 3rd
- 4th
- 5th-6th
- 7th +

37. What is your current major?

- Computer Science
- Electrical Engineering
- Computer Engineering
- Management Information Systems
- Math
- Other _____

38. What is your current GPA?

- First semester
- < 1.0
- 1.0 - 2.0
- 2.0 - 3.0
- 3.0- 4.0

39. What is your official class status?

- Senior
- Junior
- Sophomore
- Freshman
- Other

40. Are you a transfer student (i. e. did you start your freshman year somewhere other than The University of Alabama)?

- Yes
- No

41. Are you currently taking CS150?

- Yes
- No
- Dropped it
- Completed

42. Is CS150 your first programming class?

- Yes
- No - High school course
- No - Other College course
- Not taking CS150

43. What grade do you expect in CS150?

- A+, A, A-
- B+, B, B-
- C+, C, C-
- D+, D, D-
- F
- Not taking CS150

44. Do you need another programming class to graduate?

- Yes
- No

45. When do you plan to take the next programming course?

- Next semester
- Another semester
- Never

46. Gender

- Male
- Female

Demographics (2nd and 3rd Assessments)

32. At this point in the semester, how would you rate your programming skill in comparison to others in your class?

- I have a lot more skill
- I have somewhat more skill
- I have average skill
- I have somewhat less skill
- I have a lot less skill

33. At this point in the semester, how would you rate your computer knowledge in comparison to others in your class?

- I have a lot more knowledge
- I have somewhat more knowledge
- I have average knowledge
- I have somewhat less knowledge
- I have a lot less knowledge

34. Are you still intimidated by programming?

- Yes
- No

35. Are you still intimidated by Computer Science?

- Yes
- No

36. What grade do you expect in CS150?

- A+, A, A-
- B+, B, B-
- C+, C, C-
- D+, D, D-
- F
- Not taking CS150

37. Gender

- Male
- Female

PENNINGTON'S MODEL – VERSION 1 (1st and 2nd Assessments)

```
#Python program version 1
from math import *
def main():
    amount = eval (input("Enter an amount of change from 1 to 99
                        cents. "))
    quarters = amount / 25
    amountLeft = amount % 25
    dimes = amountLeft / 10
    amountLeft = amountLeft % 10
    nickels = amountLeft / 5
    amountLeft = amountLeft % 5
    pennies = amountLeft

    print ("There are ", quarters , "Quarters, ",dimes ,"Dimes, ",
           nickels ,"Nickels, and ",pennies ,"Pennies " )
main()
```

- 1: Is the variable pennies initialized to 0?[Yes / No]
2. Is the number of quarters needed calculated before the number of dimes needed?..... [Yes / No]
3. Will the value of amountLeft affect the value of pennies?[Yes / No]
4. Does amountLeft have a value before quarters is assigned a value?[Yes / No]
5. Does this program compute how to give change in the largest possible denominations? [Yes / No]

PENNINGTON'S MODEL – VERSION 2 (1st and 2nd Assessments)

```
#Python program version 2
from math import *
def main():
    amount = eval (input("Enter an amount of change from 1 to 99
                        cents. "))
    quarters = amount / 25
    amountLeft = amount % 25
    dimes = amountLeft / 10
    amountLeft = amountLeft % 10
    nickels = amountLeft / 5
    amountLeft = amountLeft % 5
    pennies = amountLeft

    print ("There are ", quarters , "Quarters, ",dimes ,"Dimes, ",
           nickels ,"Nickels, and ",pennies ,"Pennies " )
main()
```

1. Is the variable Pennies initialized to 25?[Yes / No]
2. Is the number of quarters needed calculated after the number of dimes needed?[Yes / No]
3. Will the value of AmountLeft affect the value of Nickels?.....[Yes / No]
4. Does AmountLeft have a value after Quarters is assigned a value?.....[Yes / No]
5. Does this program compute how to give change in the largest possible denominations?..[Yes / No]

Protocol Analysis - Instructions

Write a program that converts 700 days into y years, m months, and d days remaining. Use your new environment from CS 150 to complete this task.

Protocol Analysis – Instructions (with example code)

Write a program that converts 700 days into y years, m months, and d days remaining. Use your new environment from CS 150 to complete this task. Below is an example program that converts 75 minutes into h hours and m minutes remaining.

Code:

```
def main():
    duration_minutes = 75
    print('Hours')           #prints a label for value
    print(duration_minutes//60) #prints value after math is completed
    print('Minutes')        #prints a label for value
    print(duration_minutes%60) #prints value after math is completed
main()
```


IDLE - Usability (1st Assessment)

1.) What was your initial response to using IDLE to program?

1a.) Why? What factors led you to this?

2.) What is the easiest thing about using IDLE to program?

3.) What is the hardest thing about using IDLE to program?

4.) How comfortable are you with using IDLE for programming?

- 1 = "not comfortable at all" 4 = "50/50" 5 = "fairly comfortable"
 2 = "mostly not comfortable" 6 = "mostly comfortable"
 3 = "slightly comfortable" 7 = "absolutely comfortable"

5.) How often did you make a mistake or mishandle IDLE when programming?

- 1 = "absolutely often" 4 = "50/50" 5 = "slightly often"
 2 = "mostly often" 6 = "mostly NOT often"
 3 = "fairly often" 7 = "absolutely NOT often"

6.) If you were asked to complete another program assignment, how confident are you in writing one using IDLE at this point?

- 1 = "not confident at all" 4 = "50/50" 5 = "fairly confident"
 2 = "mostly not confident" 6 = "mostly confident"
 3 = "slightly confident" 7 = "absolutely confident"

7.) How much do you like IDLE after using it?

- 1 = "not at all" 4 = "50/50" 5 = "fairly like"
 2 = "mostly do not like" 6 = "mostly like"
 3 = "slightly like" 7 = "absolutely like"

7a.) Why? What factors led you to this decision?

8.) Are you currently using another programming environment?

9.) What "other" programming environment(s) are you using?

10.) Is the "other" programming environment mandatory for a course you are taking?

11.) How much do you enjoy using this "other" programming environment?

- 1 = "not at all" 4 = "50/50" 5 = "fairly like"
 2 = "mostly do not like" 6 = "mostly like"
 3 = "slightly like" 7 = "absolutely like"

12.) Between IDLE and your "other" environment, which environment do you like better?

63a.) Why? What factors led you to this?

13.) Would you use your "other" programming environment for random projects outside of a course? Why or why not?

14.) Would you use IDLE for random projects outside of a course? Why or why not?

Thank You!

Thank you for your participation

IDLE - Usability (2nd and 3rd Assessments)

1.) At this point in the semester, how comfortable are you with using IDLE for programming?

- 1 = "not comfortable at all" 4 = "50/50" 5 = "fairly comfortable"
 2 = "mostly not comfortable" 6 = "mostly comfortable"
 3 = "slightly comfortable" 7 = "absolutely comfortable"

2.) How long did it take to get comfortable with using IDLE?

- 1 = "still not comfortable" 4 = "1 month" 5 = "2 to 3 weeks"
 2 = "2 months"
 3 = "1.5 months" 6 = "1 week or less"
 7 = "already knew how to use it"

3.) At this point in the semester, how often do you make a mistake or mishandle IDLE when programming?

- 1 = "absolutely often" 4 = "50/50" 5 = "slightly often"
 2 = "mostly often" 6 = "mostly NOT often"
 3 = "fairly often" 7 = "absolutely NOT often"

4.) If you were asked to complete a program assignment today, how confident are you in writing one using IDLE at this point in the semester?

- 1 = "not confident at all" 4 = "50/50" 5 = "fairly confident"
 2 = "mostly not confident" 6 = "mostly confident"
 3 = "slightly confident" 7 = "absolutely confident"

5.) At this point in the semester, how much do you like using IDLE?

- 1 = "not at all" 4 = "50/50" 5 = "fairly like"
 2 = "mostly do not like" 6 = "mostly like"
 3 = "slightly like" 7 = "absolutely like"

5a.) Why? What factors led you to this decision?

6.) On a scale of 1 – 10 (1 = not so well, 10 = very well), how well do you know how to use IDLE?

7.) Are you currently using another programming environment?

8.) When using IDLE, do you find yourself sometimes referring back to (or using) this “other” environment to complete tasks?

- 1 = "not at all" 4 = "50/50" 5 = "fairly yes"
 2 = "mostly no" 6 = "mostly yes"
 3 = "slightly yes" 7 = "absolutely yes"

9.) What “other” programming environment(s) are you using?

10.) Is the “other” programming environment mandatory for a course you are taking?

11.) How comfortable are you with using this “other” environment for programming?

- 1 = "not comfortable at all" 4 = "50/50" 5 = "fairly comfortable"
 2 = "mostly not comfortable" 6 = "mostly comfortable"
 3 = "slightly comfortable" 7 = "absolutely comfortable"

****There are more questions on the back...***

12.) How much do you enjoy using this “other” programming environment?

- 1 = "not at all" 4 = "50/50" 5 = "fairly like"
 2 = "mostly do not like" 6 = "mostly like"
 3 = "slightly like" 7 = "absolutely like"

13.) How long did it take to get comfortable with using this “other” environment?

- 1 = "more than 2 months" 4 = "1 month" 5 = "2 to 3 weeks"
 2 = "2 months" 6 = "1 week or less"
 3 = "1.5 months" 7 = "less than 1 week"

13a.) If you chose “more than 2 months”, please indicate below the exact amount of months, years, etc.

14.) How often do you make a mistake or mishandle this “other” environment when programming?

- 1 = "absolutely often" 4 = "50/50" 5 = "slightly often"
 2 = "mostly often" 6 = "mostly NOT often"
 3 = "fairly often" 7 = "absolutely NOT often"

15.) On a scale of 1 – 10 (1 = not so well, 10 = very well), how well do you know how to use this “other” environment?

16.) When using this “other” environment(s), do you find yourself sometimes referring back to (or using) IDLE to complete tasks?

- 1 = "not at all" 4 = "50/50" 5 = "fairly yes"
 2 = "mostly no" 6 = "mostly yes"
 3 = "slightly yes" 7 = "absolutely yes"

17.) Between IDLE and your “other” environment, which environment do you like better?

17a.) Why? What factors led you to this?

18.) Would you use your “other” programming environment for random projects outside of a course? Why or why not?

19.) Would you use IDLE for random projects outside of a course? Why or why not?

Thank You!

Thank you for your participation

VIM - Usability (1st Assessment)

1.) What was your initial response to using VI/VIM to program?

1a.) Why? What factors led you to this?

2.) What is the easiest thing about using VI/VIM to program?

3.) What is the hardest thing about using VI/VIM to program?

4.) How comfortable are you with using VI/VIM for programming?

- 1 = "not comfortable at all" 4 = "50/50" 5 = "fairly comfortable"
 2 = "mostly not comfortable" 6 = "mostly comfortable"
 3 = "slightly comfortable" 7 = "absolutely comfortable"

5.) How often did you make a mistake or mishandle VI/VIM when programming?

- 1 = "absolutely often" 4 = "50/50" 5 = "slightly often"
 2 = "mostly often" 6 = "mostly NOT often"
 3 = "fairly often" 7 = "absolutely NOT often"

6.) If you were asked to complete another program assignment, how confident are you in writing one using VI/VIM at this point?

- 1 = "not confident at all" 4 = "50/50" 5 = "fairly confident"
 2 = "mostly not confident" 6 = "mostly confident"
 3 = "slightly confident" 7 = "absolutely confident"

7.) How much do you like VI/VIM after using it?

- 1 = "not at all" 4 = "50/50" 5 = "fairly like"
 2 = "mostly do not like" 6 = "mostly like"
 3 = "slightly like" 7 = "absolutely like"

7a.) Why? What factors led you to this decision?

8.) Are you currently using another programming environment?

9.) What "other" programming environment(s) are you using?

10.) Is the "other" programming environment mandatory for a course you are taking?

11.) How much do you enjoy using this "other" programming environment?

- 1 = "not at all" 4 = "50/50" 5 = "fairly like"
 2 = "mostly do not like" 6 = "mostly like"
 3 = "slightly like" 7 = "absolutely like"

12.) Between VI/VIM and your "other" environment, which environment do you like better?

63a.) Why? What factors led you to this?

13.) Would you use your "other" programming environment for random projects outside of a course? Why or why not?

14.) Would you use VI/VIM for random projects outside of a course? Why or why not?

Thank You!

Thank you for your participation

VIM - Usability (2nd and 3rd Assessment)

1.) At this point in the semester, how comfortable are you with using VIM for programming?

- 1 = "not comfortable at all" 4 = "50/50" 5 = "fairly comfortable"
 2 = "mostly not comfortable" 6 = "mostly comfortable"
 3 = "slightly comfortable" 7 = "absolutely comfortable"

2.) How long did it take to get comfortable with using VIM?

- 1 = "still not comfortable" 4 = "1 month" 5 = "2 to 3 weeks"
 2 = "2 months" 6 = "1 week or less"
 3 = "1.5 months" 7 = "already knew how to use it"

3.) At this point in the semester, how often do you make a mistake or mishandle VIM when programming?

- 1 = "absolutely often" 4 = "50/50" 5 = "slightly often"
 2 = "mostly often" 6 = "mostly NOT often"
 3 = "fairly often" 7 = "absolutely NOT often"

4.) If you were asked to complete a program assignment today, how confident are you in writing one using VIM at this point in the semester?

- 1 = "not confident at all" 4 = "50/50" 5 = "fairly confident"
 2 = "mostly not confident" 6 = "mostly confident"
 3 = "slightly confident" 7 = "absolutely confident"

5.) At this point in the semester, how much do you like using VIM?

- 1 = "not at all" 4 = "50/50" 5 = "fairly like"
 2 = "mostly do not like" 6 = "mostly like"
 3 = "slightly like" 7 = "absolutely like"

5a.) Why? What factors led you to this decision?

6.) On a scale of 1 – 10 (1 = not so well, 10 = very well), how well do you know how to use VIM?

7.) Are you currently using another programming environment?

8.) When using VIM, do you find yourself sometimes referring back to (or using) this “other” environment to complete tasks?

- 1 = "not at all" 4 = "50/50" 5 = "fairly yes"
 2 = "mostly no" 6 = "mostly yes"
 3 = "slightly yes" 7 = "absolutely yes"

9.) What “other” programming environment(s) are you using?

10.) Is the “other” programming environment mandatory for a course you are taking?

11.) How comfortable are you with using this “other” environment for programming?

- 1 = "not comfortable at all" 4 = "50/50" 5 = "fairly comfortable"
 2 = "mostly not comfortable" 6 = "mostly comfortable"
 3 = "slightly comfortable" 7 = "absolutely comfortable"

**There are more questions on the back...*

12.) How much do you enjoy using this “other” programming environment?

- 1 = "not at all" 4 = "50/50" 5 = "fairly like"
 2 = "mostly do not like" 6 = "mostly like"
 3 = "slightly like" 7 = "absolutely like"

13.) How long did it take to get comfortable with using this “other” environment?

- 1 = "more than 2 months" 4 = "1 month" 5 = "2 to 3 weeks"
 2 = "2 months" 6 = "1 week or less"
 3 = "1.5 months" 7 = "less than 1 week"

13a.) If you chose “more than 2 months”, please indicate below the exact amount of months, years, etc.

14.) How often do you make a mistake or mishandle this “other” environment when programming?

- 1 = "absolutely often" 4 = "50/50" 5 = "slightly often"
 2 = "mostly often" 6 = "mostly NOT often"
 3 = "fairly often" 7 = "absolutely NOT often"

15.) On a scale of 1 – 10 (1 = not so well, 10 = very well), how well do you know how to use this “other” environment?

16.) When using this “other” environment(s), do you find yourself sometimes referring back to (or using) VIM to complete tasks?

- 1 = "not at all" 4 = "50/50" 5 = "fairly yes"
 2 = "mostly no" 6 = "mostly yes"
 3 = "slightly yes" 7 = "absolutely yes"

17.) Between VIM and your “other” environment, which environment do you like better?

17a.) Why? What factors led you to this?

18.) Would you use your “other” programming environment for random projects outside of a course? Why or why not?

19.) Would you use VIM for random projects outside of a course? Why or why not?

Thank You!

Thank you for your participation

APPENDIX F: IRB CERTIFICATE

This appendix contains an IRB approval certificate for this research.



Certificate of Completion

The National Institutes of Health (NIH) Office of Extramural Research certifies that **Edward Dillon** successfully completed the NIH Web-based training course "Protecting Human Research Participants".

Date of completion: 02/14/2011

Certification Number: 633250