A Meta-Analysis of Pedagogical Tools used in Introductory Programming Courses

by

Frances P. Trees

Submitted in partial fulfillment
of the requirements for the degree of
Doctor of Professional Studies
in Computing

at

Seidenberg School of Computer Science and Information Systems

Pace University

March 2010

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

We hereby certify that this dissertation, submitted by Frances P. Trees, satisfies the dissertation requirements for the fegree of Doctor of Professional Studies in Computing has been approved.

Dr. Joe Bergin

Dr. Fred Grossman

Dr. Charles Tappert

School of Computer Science and Information Systems
Pace University 2010

Dedication

This dissertation is dedicated to the computer science teachers who work to make education better and more accessible for all students and to the tool developers who continue to offer computer science teachers new and innovative pedagogical tools to add to their teaching repertoire.

Preface

Disclaimers:

The researcher acknowledges professional contacts with many of the developers of the tools investigated in this study.

- Joe Bergin, an author of *Karel J. Robot*, is the researcher's doctorate advisor and chair of the dissertation committee.
- The researcher has served on the Board of Directors of the Computer Science Teachers Association (CSTA) with Steve Cooper, a member of the Alice development team.
- The researcher is a member of the College Board Advanced Placement Computer Science (AP CS) Test Development Committee. This committee is the author of the Microworld, GridWorld.
- Barb Ericson, developer of the Media Computation Library, serves on the CSTA Board of Directors and the AP CS Test Development Committee with the researcher.
- The researcher was a member of the Java Task Force, working with Eric Roberts, the primary developer of ACM's Java Task Force Graphics Library. Members of this task force also included James Cross, developer of the jGRASP IDE, Kim Bruce, developer of the ObjectDraw Library, and Ian Utting, member of the BlueJ and Greenfoot development teams.
- The researcher has served on the AP CS Test Development Committee and a College Board Ad Hoc Committee for AP CS with Corky Cartwright, a developer of DrJava.

The researcher has not been influenced by any of these relationships in the development or presentation of this research study.

Acknowledgements

I wish to thank my advisor, Dr. Joseph Bergin, and my advisory committee, Dr. Fred Grossman and Dr. Charles Tappert, whose guidance and support has helped me to become a better researcher.

I would like to acknowledge all of the tool developers that have contributed so much to improving computer science education. You have provided a means to improve and enhance the teaching and learning of programming.

I thank the computer science educators and researchers who helped with the development of the survey and the teachers who contributed their valuable time to complete the survey.

I also would like to thank those teachers and tool developers who agreed to be interviewed for this study. It was a pleasure and an honor talking with you. Your professional opinions and input to this study were invaluable.

And finally, I would like to thank Eli Tepperberg. You were my support throughout the program and through this dissertation (even though it interfered with all of our play time).

An Abstract

A Meta-Analysis of Pedagogical Tools
Used in Introductory Programming Courses

by
Frances P. Trees

Submitted in partial fulfillment
of the requirements for the degree of
Doctor of Professional Studies
in Computing

March 2010

Programming is recognized as being challenging for teachers to teach and difficult for students to learn. For decades, computer science educators have looked at innovative approaches by creating pedagogical software tools that attempt to facilitate both the teaching of and the learning of programming. This dissertation investigates the motivations for the integration of pedagogical tools in introductory programming courses and the characteristics that are perceived to contribute to the effectiveness of these tools.

The study employs three research stages that examine the tool characteristics and their use. The first stage surveys teachers who use pedagogical tools in an introductory programming course. The second interviews teachers to explore the survey results in more detail and to add greater depth into the choice and use of pedagogical tools in the introductory programming class. The third interviews tool developers to provide an explanatory insight of the tool and the motivation for its creation.

The results indicate that the pedagogical tools perceived to be effective share common characteristics: They provide an environment that is manageable, flexible and visual; they provide for active engagement in learning activities and support programming in small pieces; they allow for an easy transition to subsequent courses and more robust environments; they provide technical support and resource materials. The results of this study also indicate that recommendations from other computer science educators have a strong impact on a teacher's initial tool choice for an introductory programming course.

This study informs present and future tool developers of the characteristics that the teachers perceive to contribute to the effectiveness of a pedagogical tool and how to

present their tools to encourage a more efficient and more effective widespread adoption of the tool into the teacher's curriculum.

The teachers involved in this study are actively involved in the computer science education community. The results of this study, based on the perceptions of these computer science educators, provide guidance to those educators choosing to introduce a new pedagogical tool into their programming course.

Table of Contents

Dedication . . . . . **iii**

Preface . . . . . **iv**

Acknowledgements . . . . . **v**

An Abstract . . . . . **vi**

List of Tables . . . . . **xiii**

List of Figures . . . . . **xvii**

Chapters

1 Overview **1**

2 Introduction **4**

2.1 Overview 4

2.2 A Change in Pedagogy 7

2.3 Road Map of the Dissertation 9

3 Addressing the Use of Pedagogical Tools **11**

3.1 Pedagogical Tools Overview 11

3.2 Teaching 16

3.3 Learning 17

3.4 The Importance of Teacher Perceptions 19

3.5 Research Questions 20

3.6 Limitations of the Study 24

3.7 Significance of the Study 24

4 Teaching and Learning: An Overview **27**

4.1 Introduction 27

4.2 Learning Styles 28

4.3 Learning Styles and Teaching Students 35

5 Pedagogical Tools: Educational Research . . . 39
5.1 Visualization Tools . . . 41
5.2 Microworlds . . . 44
5.3 Robots . . . 47
5.4 IDEs . . . 48
5.5 Games . . . 52
5.6 Libraries . . . 54
5.7 Summary . . . 59
6 Research Methodology . . . 62
6.1 Introduction . . . 62
6.2 Research Process Design . . . 63
6.3 Stage 1: Survey to Teachers . . . 66
6.3.1 Introduction . . . 66
6.3.2 Survey Design and Development . . . 67
6.3.3 Subject Privacy and Confidentiality . . . 69
6.3.4 Sampling and Subject Selection . . . 70
6.3.5 Distribution and Response Management . . . 71
6.3.6 Survey Piloting . . . 72
6.4 Stage 2: Interviews with Teachers . . . 73
6.4.1 Introduction . . . 73
6.4.2 Interview Design and Development . . . 73
6.4.3 Subject Privacy and Confidentiality . . . 74
6.4.4 Sampling and Subject Selection . . . 74
6.4.5 Interview Process and Response Management . . . 75
6.5 Stage 3: Interviews with Tool Developers . . . 76
6.5.1 Introduction . . . 76
6.5.2 Interview Design and Development . . . 76
6.5.3 Subject Privacy and Confidentiality . . . 77

6.5.4 Sampling and Subject Selection 77
6.5.5 Interview Process and Response Management 77
6.6 Summary 78

7 Results 79

7.1 Introduction 79
7.2 Teacher Survey Results 80
7.2.1 Survey Question Types 84
7.2.2 Analysis of Rank Data 86
7.2.3 Cross Categories Summary 87
7.2.3.1 Choosing to Use the Tool 88
7.2.3.2 Overall Tool Characteristics 89
7.2.3.3 Programming Environment, Testing, Debugging, and Interaction 93
7.2.3.4 Learning and Teaching 98
7.2.3.5 Auxiliary Materials and Support 101
7.2.3.6 Negative Aspects of Tool 106
7.2.4 Aggregation: Summary of the Individual Tool Categories . . 108
7.2.4.1 Choosing to Use the Tool 109
7.2.4.2 Overall Tool Characteristics 110
7.2.4.3 Programming Environment, Testing, Debugging, and Interaction 112
7.2.4.4 Learning and Teaching 116
7.2.4.5 Auxiliary and Support 119
7.2.4.6 Negative Aspects of Tool 122
7.2.5 Training and Experience 123
7.2.6 Course Description and Teacher Experience 130
7.2.6.1 Course Description 130
7.2.6.2 Teacher Demographics and Experience 131
7.3 Interviews with Teachers 132

7.4 Interviews with Tool Developers 139

8 Analysis of Results 143

8.1 Introduction 143

8.2 Analysis Presentation 143

8.3 Addressing the Research Questions 145

8.3.1 What Influences the Use of Pedagogical Tools in the Introductory Programming Class? 146

8.3.1.1 Tool Choice 146

8.3.1.2 Recommendations from Others 147

8.3.1.3 Teaching Experience 150

8.3.1.4 Learning the Tool 150

8.3.2 What Characteristics are Perceived by Teachers to Contribute to the Effectiveness of the Tool(s) they Choose to Use in an Introductory Programming Course? 152

8.3.2.1 Manageable Environment 153

8.3.2.2 Active Learning 155

8.3.2.3 Visual Environment 156

8.3.2.4 Flexible Environment 156

8.3.2.5 Subsequent Courses 157

8.3.2.6 Programming 160

8.3.2.7 Tool Resources 161

8.3.2.8 Teaching 163

8.3.2.9 Learning 164

8.3.3 What are the Perceived Characteristics of a Pedagogical Tool that Hinder (or get in the way of) Teaching and/or Learning in an Introductory Programming Course? 165

9 Summary and Future Work 167

9.1 Addressing the Hypotheses 168

9.1.1 Pedagogical Tools and the Learning of Programming 168

9.1.2 Pedagogical Tools and the Teaching of Programming 169

9.1.3 Characteristics of Tools Used in Introductory Programming Classes . . . 170
9.1.4 Characteristics of Tools NOT Used in Introductory Programming Classes . . . 172
9.1.5 Choosing to Use a Pedagogical Tool . . . 173
9.1.5.1 Information for Tool Developers and Teachers . . . 173
9.2 Related Work and Research . . . 176
9.3 Contribution to Knowledge . . . 177
9.3.1 Information for the Tool Developers . . . 178
9.3.2 Information for the Teachers . . . 179
9.4 Future Work . . . 179

Appendices

Appendix A — Pedagogical Tools Survey for Teachers . . . 182
Appendix B — Survey Letters . . . 203
Appendix C — Interview Questions . . . 207
Appendix D — Developers' Goals and Tool Characteristics . . . 210
Appendix E — Themes . . . 216

List of Tables

1 Mapping of Developers' Goals with Survey Questions (example) . . . 68

2 Tools evaluated through this survey by teachers of introductory programming classes . . . 81

3 Tools chosen to be used or chosen not to be used by teachers . . . 83

4 Top three characteristics, ranked in priority order, that are perceived to contribute to the tool's effectiveness as determined by the total survey responses, the responses of the users, and the responses of the non-users . . . 90

5 Top three characteristics, ranked in priority order, that are perceived to contribute to the tool's effectiveness as determined by the total survey responses, the male responses, and the female responses . . . 91

6 Characteristics, in priority order, that are perceived to be inadequately supported by the tool as determined by the total survey responses, the responses of the users, and the responses of the non-users . . . 92

7 Characteristics, in priority order, perceived to be inadequately supported by the tool as determined by the total survey responses, the responses of the males, and the responses of the females . . . 92

8 Characteristics, in priority order, perceived to be inadequately supported by the tool as determined by the total survey responses, the responses of the college teachers, and the responses of the secondary school teachers 93

9 Top three ranked characteristics, in priority order, that relate to testing, debugging, and interaction that contribute to the effectiveness of this tool as ranked by the total survey responses, the responses of the users, and the responses of the non-users . . . 96

10 Top three ranked characteristics, in priority order, that relate to testing, debugging, and interaction that are perceived to contribute to the effectiveness of this tool as ranked by the total survey responses, male responses, and female responses . . . 96

11 Top three characteristics that relate to testing, debugging, and interaction that are perceived to contribute to the effectiveness of this tool as ranked by total survey responses, responses of college teachers, and responses of secondary school teachers . . . 97

12 Characteristics, in priority order, that are perceived to be inadequately supported by the tool as determined by the total survey responses, the responses of the users, and the responses of the non-users . . . 98

13 Characteristics, in priority order, that are perceived to be inadequately supported by the tool as determined by the total survey responses, the male responses, and the female responses 98

14 Characteristics relating to programming environment, debugging, testing, and interaction, in priority order, that are perceived to be inadequately supported by the tool as ranked by total survey responses, the responses of the college teachers, and the responses of the secondary school teachers 99

15 Characteristics that are perceived to support the learning and teaching of programming as ranked, in priority order, by the total survey responses, the responses of the users, and the responses of the non-users 101

16 Characteristics that are perceived to support the learning and teaching of programming as ranked, in priority order, by the total survey responses, the male responses, and the female respondents 102

17 Characteristics that are perceived to support the learning to and teaching of programming as ranked, in priority order, by the total responses, the responses of the college teachers, and the responses of the secondary school teachers 102

18 Characteristics dealing with auxiliary materials and support that are perceived to contribute to the effectiveness of the tool as ranked by total survey responses, the responses of the users, and the responses of the non-users 104

19 Characteristics dealing with auxiliary materials and support that are perceived to contribute to the effectiveness of the tool as ranked by total survey responses, the male responses, and the female responses 105

20 Characteristics relating to auxiliary material and support, in priority order, that are perceived to be inadequately supported by the tool as ranked by total survey responses, the responses of the users, and the responses of the non-users 105

21 Characteristics relating to auxiliary material and support, in priority order, that are perceived to be inadequately supported by the tool as ranked by total survey responses, the male responses, and the female responses 106

22 Characteristics relating to auxiliary material and support, in priority order, that are perceived to be inadequately supported by the tool as ranked by total survey responses, the responses of the college teachers, and the responses of the secondary school teachers 107

23 Reasons for initially choosing to use this tool in an introductory programming course chosen by a majority of the responses in each tool category . . 110

24 Characteristics relating to user interface and student interaction perceived to contribute to the effectiveness of the tool by more than 50% of the responses in each tool category 111

25 Top three ranked characteristics, in priority order, that are perceived to contribute to the tool's effectiveness as determined by responses for each tool category 112

26 Top three characteristics, in priority order, that are perceived to be inadequately supported by the tool as determined by the survey responses evaluating Microworlds and IDEs 113

27 Characteristics related to programming environment perceived to contribute to the effectiveness of the tool by more than 50% of those repsonding within each tool category 113

28 Characteristics related to testing, debugging, and interaction perceived to contribute to the effectiveness of the tool by the majority of the responses in each tool category 114

29 Top three ranked characteristics, in priority order, that relate to the programming environment, testing, debugging, and interaction that are perceived to contribute to the effectiveness of this tool as determined by responses for each tool category 115

30 Characteristics, in priority order, that are perceived to be inadequately supported by the tool as determined by responses in each tool category 116

31 The characteristics that are perceived to contribute to student learning identified by more than 50% of the responses in each tool category . 117

32 The characteristics related to the teaching of programming that are perceived to contribute to effective teaching identified by more than 50% of the responses in each tool category 118

33 Characteristics, in priority order, that are perceived to contribute to the effectiveness of the tool as related to the learning of and teaching of programming as determined by responses for each tool category . 118

34 The characteristics related to auxiliary materials and support that are perceived to contribute to effective teaching identified by more than 50% of the responses in each tool category 119

35 The characteristics related to the overall perception of the tool identified by more than 50% of the responses in each tool category 120

36 Top three characteristics, in priority order, that are perceived to contribute to the effectiveness of the tool as related to auxiliary materials, support, and overall perceptions of the tool as determined by responses for each tool category 121

37 Characteristics related to auxiliary materials and support, listed in priority order, that are perceived to be inadequately supported by the tool as determined by responses for three tool categories 121

38 Characteristics related to the tool environment that are perceived to hinder teaching or learning by those survey responses evaluating Microworlds and IDEs 123

39 The characteristics related to errors and support that are perceived to hinder teaching or learning as determined by responses for each tool category 124

40 Type of training received in the use of the tool for each tool category 125

41 Descriptive statistics (frequency, median, mode) for the total survey responses 126

42 Descriptive statistics (frequency, median, mode) for the responses of those presently using the tool 127

43 Descriptive statistics (mean and standard deviation) for the total survey responses 128

44 Descriptive statistics (mean and standard deviation) for the responses of those presently using the tool 129

45 Mapping of Developers' Goals with Tool Characteristics from Survey Questions 215

List of Figures

1 A Simplified Version of the Kolb's Learning Cycle 31

2 Based on Kolb's Learning Cycle with Identified Quadrants 32

3 Graphical Illustration of Mixed Research Design (based on the illustrations from Teddlie and Tashakkori) [212] 65

Chapter 1

Overview

Today's programming students have grown up in a world of emerging technologies. They interact with information differently from previous generations. Researchers in the field of computer science education have developed tools that help programming teachers address the needs and learning styles of today's students. These pedagogical tools facilitate the teaching of and the learning of programming.

This research investigates the characteristics that contribute to the effectiveness of pedagogical tools used in introductory programming courses. More specifically, this research determines those characteristics that are common to effective pedagogical tools used in introductory programming courses. Where past research examined a specific tool or a specific category of tools (Microworlds, Libraries, Integrated Development Environments, Visualization tools, etc.), this research looks at characteristics that are common to all tools, regardless of category. The tool characteristics investigated in this study are classified under the following major themes:

- Manageable Environment: The tool was not difficult to use or install, simplified the mechanics of programming, and was neither too restrictive or too complicated for an introductory programming course.
- Active Learning: The tool supports an interactive environment where students are actively engaged in the learning process.
- Good First Experience: Students enjoy using the tool and programming is introduced in an enjoyable way through the use of the tool.

- Visual Environment: The tool supports some form of graphical components or visualization techniques.
- Flexible Environment: The tool engages many levels of learning and can be used throughout the course.
- Subsequent Courses: The tool is a solid introduction to subsequent computer science courses and allows students to transition to these courses seamlessly.
- Programming Activities: The tool provides support for programming activities.
- Tool Resources: The developer, tool, or community, provides resources for using the tool in an introductory programming course.
- Teaching: The tool eases and promotes the teaching of programming.
- Learning: The tool eases and promotes the learning of programming.

The results indicate that an effective pedagogical tool provides opportunities for active learning and provides visual representations that facilitate the understanding of programming concepts and program execution. An effective tool allows for a flexible programming environment that addresses the needs of students of various ability levels and learning styles and provides learning experiences that encourage students to continue in the discipline with a seamless transitioning experience.

This study also seeks to determine the greatest influence in initially adopting a tool to integrate into the introductory programming curriculum and the reasons the teacher initially chooses one tool over another. The results indicate that recommendations from other computer science educators have a strong impact on a teacher's initial tool choice for an introductory programming course.

The results of this study, when communicated to other computer science educators, provide information that may serve as a guide in the adoption of pedagogical tools in an introductory programming course. The results also inform present and future tool developers of the characteristics that the teachers perceive to contribute to the effectiveness of a pedagogical tool and how to present their tools to encourage a more efficient and more effective widespread adoption of the tool into the teacher's curriculum.

The study employs three research stages that examine the tool characteristics and their use. The first stage surveys teachers who use pedagogical tools in an introductory programming course. The second interviews teachers to explore the survey results in more detail and to add greater depth into the choice and use of pedagogical tools in the introductory programming class. The third interviews tool developers to provide an explanatory insight of the tool and the motivation for its creation.

The teachers involved in this study are actively involved in the computer science education community. The results of this study, based on the perceptions of these computer science educators, provide guidance to those educators choosing to introduce a new pedagogical tool into their programming course and information to those tool developers who are developing new tools or modifying existing tools.

Chapter 2

Introduction

2.1 Overview

Although programming is a key objective in most introductory computing classes, it is a skill that is both challenging for teachers to teach [91, 94, 124, 140, 200, 231] and difficult for students to learn [18, 91, 94, 116, 149, 151, 177, 214, 225]. This dissertation investigates the motivations for the integration of pedagogical tools in introductory programming courses and the characteristics of these tools.

Past research on predictors of success and failure in introductory computer science courses [19, 20, 216, 226, 227] identifies factors that could certainly help educators *select* students that have the best chances for success. *Selection* may not be a luxury of educators today. According to the Computing Research Association (CRA), the number of new computer science majors in 2004 was 40% lower than in 2000 and the percentage of incoming undergraduates among all degree-granting institutions who indicated they would major in computer science declined by 70% between fall 2000 and 2005 [219]. Although the 2006 - 2007 CRA Taulbee Survey boasts an 18% increase in the number of Ph.D.s awarded between July 2006 and June 2007, the undergraduate enrollments remain unclear. The Bachelor's degree production was down 20% in 2006 - 2007 and the fraction of Bachelor's degrees granted to women dropped from 14.2% to 11.8% with many programs reporting less than 10% female enrollment. Enthnicity is also less diverse, with White, non-Hispanics receiving approximately 66% (up from 59.6% in 2006) of the Bachelor's degrees being granted in 2007 [208].

In March 2009, The NY Times [148], The Seattle Post [102], and multitudes of

higher education and technical news reports boasted that for the first time in six years, enrollment in computer science programs in the United States increased. The number of majors and pre-majors in American computer science programs was up 6.2% from 2007. The reports referenced the most recent (2007 - 2008) CRA Taulbee Survey in which a total of 264 Ph.D.-granting departments were surveyed. It should be noted that Taulbee Surveys of previous years reported on departments of computer science. Information Schools (I-Schools) that support Information (I) programs such as Information Science, Information Systems, Information Technology, and Informatics, were not previously included in Taulbee Survey results. The 2007-2008 Taulbee Survey was the first time that the count included Information-school departments. 19 of the 264 departments surveyed were I-School departments.

Although the total Ph.D. production among the responding departments represents a 5.7% increase in 2008, Bachelor's degree production in computer science was down 10%, compared to a nearly 20% decline in the previous year. Diversity in computer science undergraduate programs remains poor. The fraction of Bachelors degrees awarded to women held steady at 11.8%. As was the case in 2007, nearly two-thirds of those receiving Bachelors degrees were White, non-Hispanics [27].

Enrollment trends and statistics suggest that research in computer science education might benefit by a change in the focus of the research. Where research in the past focused on predictors of success that might be used to *weed out* the students that would not succeed in computer science courses [29], those same predictors can now be used to recruit and encourage students to pursue the field. In the past there were more students interested in computer science than universities could handle [91]. Today, our pool of students is smaller and more diverse. Our pedagogy needs to change to accommodate a different programming paradigm,

different learning styles, and a more diverse student population. Although the student population and pedagogy has changed and will continue to change, the challenge remains. Programming is still challenging for teachers to teach and difficult for students to learn.

This dissertation is motivated by the need to investigate changes in pedagogy that can address the challenges that teachers and students face in an introductory programming course. In particular, this study investigates the motivations for the integration of pedagogical tools in introductory programming classes and how teachers perceive the effectiveness of these tools.

The primary goal of this research is to determine the characteristics that teachers perceive to contribute to the effectiveness of the tools, thereby easing and promoting the learning and teaching of programming. A secondary goal of this research is to examine the reasons that teachers choose to use (or choose NOT to use) pedagogical tools in their introductory programming courses.

Past studies have presented anecdotal evaluations of specific tools [117, 199, 224], analytic studies that examine a tool and its conformance to a certain set of criteria [85], and empirical studies that present quantitative or qualitative results about an individual characteristic of specific tools [86, 100, 160, 184].

This research is different. Where past research examined a specific tool or a specific category of tools, this research looks at characteristics that are common to all tools, regardless of category. This research also provides an added dimension to the already existing research results by placing an emphasis on the teachers' perceptions of the pedagogical tools chosen to be integrated into introductory programming courses. Pedagogical tools are innovations developed with the goals of making programming more accessible and of easing the learning of programming

[6, 28, 108, 109]. The influence of perception on use has considerable support in the literature [146]. Perceived ease of use and perceived usefulness are characteristics that have been studied in the acceptance of technological innovations for the past three decades [81, 146, 195]. This research investigates the perceptions that programing teachers have about pedagogical tools chosen to be used in their classrooms. The purpose of this research is twofold. First, the results inform present and future tool developers of the characteristics that are important to the teacher and how to present their tools to encourage a more efficient and more effective widespread adoption of the tool into the teacher's curriculum. Second, the results offer information to teachers looking to adopt a pedagogical tool in an introductory programming class.

The perceptions gathered in this study are those of computer science educators that are actively involved in the computer science education community. They are most likely at a high level of involvement in professional conferences, discussion groups, and professional development activities.

2.2 A Change in Pedagogy

Teachers of introductory programming courses are confronted not only with decreasing enrollments but also with declining retention rates of the students that do enroll in their courses. In more recent years the blame has fallen on the trend towards object-oriented programming [3, 32, 50, 138, 141, 220]. The language shift to Java sparked a discussion on the Special Interest Group on Computer Science Education (SIGCSE) mailing list that raised issues on how to teach this programming paradigm. Discussions on objects early verses objects late led to a reaction of educators who have successfully implemented the objects-early approach with the use of software tools or libraries. These educators encouraged other faculty to experiment with pedagogical IDEs, special libraries providing useful classes, or

microworlds [32]. Although there was insufficient data to evaluate the effectiveness of the tools in teaching introductory courses, the perception of being successful played an important role in the instructors use and promotion of the individual pedagogical tools. Kim Bruce, an author of the ObjectDraw library, concluded the summary of the SIGCSE discussion: "It would help if a group of experts in educational research were to design experiments that will allow faculty to examine the success of the innovative approaches proposed for teaching Java in CS1 [32]."

In describing the BlueJ system and its pedagogy, Michael Kölling, et.al. hypothesized that "teaching object orientation is not intrinsically more complex, but that it is made more complicated by a profound lack of appropriate tools and pedagogical experience with this paradigm and that many teachers experience serious problems when teaching object orientation but many of the problems could be overcome or reduced through the use of more appropriate tools [124]."

Since the 1970s, computer science educators have looked at innovative approaches by creating pedagogical software tools that attempt to facilitate both the teaching of and the learning of programming [231]. But the development of these tools does not in itself make teachers teach better or students learn more easily. The appropriate pedagogical tools must be chosen for the task and then the tool must be successfully integrated into the curriculum. Teachers are the most influential factor and the connecting link between the pedagogical software and the students [165, 135]. But how does the teacher learn about the tools? What types of pedagogical tools do the teachers perceive as contributing to the effectiveness in the learning of and the teaching of programming? What makes a tool *successful*? Do these *successful* tools share common characteristics?

Even though teachers are the keys to making learning happen and improving instruction, the educational research community offers little information on what

affects the teachers' adoption of educational innovations [165].

This dissertation addresses these questions with multiple sources of data in three research stages. The first surveys teachers who use pedagogical tools in an introductory programming course to identify the tools most commonly used and to explore the characteristics that are common to these tools. The second explores the survey results in more detail through interviews with teachers that use, have used, or consciously choose not to use tools in their programming courses. This provides more detailed information on the reasons for the use (and non-use) of the tools and adds depth to the survey results. Finally, to provide an explanatory insight of the tool and the motivation for its creation, interviews with tool developers of some of the more popular tools were conducted.

2.3 Road Map of the Dissertation

The organization of the remaining chapters in this dissertation is as follows:

Chapter 3 (Addressing the Use of Pedagogical Tools) discusses the motivation and purpose of this research, the research questions, the limitations and the significance of this research.

Chapter 4 (Teaching and Learning: An Overview) provides a literature review that addresses the state of computer science education, student learning styles, and teaching methodologies, as they relate to pedagogical tools and this research study.

Chapter 5 (Pedagogical Tools: Educational Research) discusses past research on the specific tools or categories of tools.

Chapter 6 (Research Methodology) identifies the methodology employed in this study and describes the three research stages that examine the characteristics and use of the pedagogical tools. The first stage surveys teachers who use pedagogical

tools in an introductory programming course. The second interviews teachers to explore the survey results in more detail and to add greater depth into the choice and use of pedagogical tools in the introductory programming class. The third interviews tool developers to provide an explanatory insight of the tool and the motivation for its creation.

Chapter 7 (Results) summarizes the data and presents the results from teacher surveys, interviews with teachers, and interviews with tool developers.

Chapter 8 (Analysis of Results) discusses the research data and suggests some explanations for the results. It explores the commonalities and differences of the tool characteristics. In addition, the comments from the respondents in stages two and three are incorporated into the analysis to provide further insight and support of the findings.

Chapter 9 (Summary and Future Work) presents responses to this study's research questions, gives the value of this research, and provides suggestions for further work in this area.

The Appendices contain the actual survey administered to the teachers, email correspondences about the survey, the interview questions posed to the teachers, the interview questions posed to the tool creators, a mapping of the tool developer's goals to the interview and survey questions, and a tabulation of the common themes that surfaced as a result of the surveys and interviews.

Chapter 3

Addressing the Use of Pedagogical Tools

3.1 Pedagogical Tools Overview

For the past five decades, researchers have developed pedagogical tools with the goals of making programming more accessible and of easing and promoting the learning of programming. While there are a considerable number of software tools that have been developed to achieve these goals, many teachers struggle with finding and evaluating the existing tools, identifying the tools that are still being maintained, and keeping up with new changes and tool developments [22]. In computer science education, pedagogical tools are designed to contribute positive experiences for novice programmers. Tool developers implement a variety of approaches in designing pedagogical tools to achieve this outcome. Most recently, tools are emerging as new features in existing tools or as the combinations of features of existing tools [178]. As examples, Storytelling Alice is a modified version of Alice that better supports girls in creating animated stories [115] and Greenfoot is a combination of object interaction (BlueJ) and object visualization (microworlds such as Karel the Robot) [82].

Research is about learning [70]. Researchers who are developing new tools can learn from the extensive collection of existing tools and the motivations that teachers have when adopting tools. This research addresses the teachers' motivations for choosing to use pedagogical tools in the introductory programming class.

Although a myriad of research projects have been devoted to tool development, only a small percentage of these tools are being successfully disseminated and implemented by the teachers in introductory programming classes. Guzdial suggests

that "the greatest contributions to be made in this field (computer science education) are not in building yet more novice programming environments but in figuring out how to study the ones we have [70]." A followup suggestion might be that the greatest contributions to be made in this field are studying the tools that we have *through the eyes of the teachers using the tools* before we build yet more new novice programming environments.

Since the teacher chooses to use (or not to use) a pedagogical tool in an introductory programming class, the developer's ability to disseminate and promote information about the tool is an important component for its widespread adoption. Palak and Walls cite numerous studies that confirm that the teachers' beliefs guide pedagogical decisions and actions taken in the classroom [169]. Their research suggests that any inquiry into classroom practices should involve an investigation of the teachers' beliefs. Teachers are key agents of change and it is the teachers' beliefs and perceptions that influence their choices in the classroom [168, 169]. This study investigates the teachers' perceptions of the characteristics that contribute to a tool's effectiveness and the motivations for integrating the tool into their curriculum.

For the purpose of this research, the tools will be classified into different categories. Included here is a brief explanation for each category in order to clarify the way in which the categories are referenced throughout this dissertation.

- Microworlds: A microworld is a learner-centered world that is explored by directly manipulating objects in the world with a limited set of simple commands [86]. More generally described by Latour, "A microworld is a tiny world inside which a student can explore alternatives, test hypotheses, and discover facts that are true about that world [128]." Microworlds provide metaphors where storytelling can occur. Examples of microworlds referenced in this study include Alice [6], Greenfoot [82], GridWorld [83], Jeroo [108], and

Karel[24].

- Visualization tools: Visualizations are graphical displays of information. Program visualizations consist of different graphical objects (often animated) and textual objects visualizing the execution of programs [184]. Visualization technology aims to help students understand how algorithms work in order to ease the problems of learning to program [126]. Examples of visualization tools referenced in this study include Jeliot 3, a program visualization application that visualizes how a Java program is interpreted [107] and jGRASP, a tool providing automatic generation of control structure diagrams for code visualization, UML class diagrams for architectural visualization, and viewers for dynamic views of primitives and objects [109]. Whereas BlueJ offers static visualization of objects, Jeliot 3 and jGRASP offer dynamic visualization techniques. RAPTOR is an iconic programming environment, designed specifically to help students visualize classes and methods and limit syntactic complexity. RAPTOR programs are created visually using a combination of UML and flowcharts [38].

- Integrated Development Environments (IDEs): An integrated development environment is a software application that provides several utilities for programmers and software developers packaged into one bundle. This bundle usually includes a source code editor, a compiler or interpreter, automated building tools, and a debugger. Most IDEs provide a way to assist students in writing correct syntax in a language in which they may not be proficient, provide a simple interface to the language compiler, flag any syntax errors, and include a mechanism for running a program [188]. Some IDEs were specifically designed for introductory programming classes providing a simple, unintimidating environment with an interactive interface [7, 28, 188].

Examples of IDEs referenced in this study are BlueJ [28], DrJava [61], DrScheme [62], Eclipse [64], Greenfoot [82], JCreator [106], jGRASP [109], and Netbeans [163].

- Robots: In the context of this study, robots refer to the LEGO Mindstorms^{TM}system. This system extends the traditional Lego bricks with a central control unit (the RCX) as well as motors and various kinds of sensors [121]. The robots can be programmed to explore an environment, detect obstacles and lights, and solve simple problems [94]. Another example cited in this study is Robotran, a multibody modeling at UCL-CEREM which is entirely based on Matlab/Simulink [193].

- Game Making Software: This study references the Game Maker development software available through YoYo Games. The software involves drag-and-drop actions to program, has a very short learning curve, and does not require prior programming experience [185]. The games include backgrounds, animated graphics, music, and sound effects [77].

- Libraries: In the context of this study, libraries refer to collections of Java tools that are designed to make programming easier for novices. These collections provide simplified features (e.g. GUI components) or provide rich collections of classes that are easily used by novices [32]. Libraries referenced in this study include ObjectDraw [166], the ACM Java Task Force Library [104], Java Power Tools [103], and the Media Computation Library [152].

- Miscellaneous: This category includes those types of tools that teachers perceive contribute to improved teaching and/or learning but do not fall into any of the above categories. Examples cited in this study include JavaBat, a free online Java programming practice site [105] and JUnit, a testing framework for the Java programming language [112].

These categories are not meant to be exhaustive or mutually exclusive. Certainly, one tool can cross over into several categories and some tools do not fit into any of the specific categories and will be treated separately. Some tools are specifically designed for a learning environment and others are developed primarily for a professional environment but are introduced in an educational setting.

For a pedagogical tool to be classified, or even perceived, as successful or unsuccessful, the tool must be used by teachers in their classrooms. In order for this to happen, the dissemination of information about the tool is important. This researcher's personal experience teaching workshops for programming teachers has confirmed that many pedagogical tools appropriate for introductory programming classes are not used because the teachers either have not heard of the tool, they do not know enough about the tool to integrate it into their curriculum, they do not know where to look for information and resources, or they simply do not have time to develop a curriculum that integrates the tool. Past studies confirm that teachers resist learning and using new methodologies [135, 184], that time constraints are blamed for teachers resisting the integration of new technologies into their curriculum [135, 231], and that teachers tend to teach what they feel most comfortable with [32].

This study investigates the teachers' perceptions of the effectiveness of pedagogical tools that they have chosen to incorporate into their programming courses and the motivation behind their choices. The study is not designed to measure effectiveness of student outcomes but rather to determine the characteristics common to those pedagogical tools that are perceived by the teachers to contribute to the effectiveness of the tool. The teachers involved in this study are subject matter experts and are the key to making learning happen. By examining their perceptions, an insight into their motivations in choosing to use certain tools and

choosing not to use others is obtained.

The use of pedagogical tools takes place in a classroom where teachers and students interact. In the computer science classroom, the learning environment should nurture the connection between the teacher and the student and ease and promote the teaching of and learning of programming. If this environment includes a pedagogical tool that was consciously adopted by the teacher, the motivations around its adoption contribute to the establishment of the learning environment. This study investigates the teachers' motivations for adopting pedagogical tools in the introductory programming environment.

3.2 Teaching

Within most educational systems internationally, the task of ensuring that pre-college teachers are adequately and appropriately prepared to teach a given discipline at a specified educational level rests with the bodies responsible for teacher certification [55]. For K-12 teachers of computer science, however, there are no consistent requirements regarding knowledge of the content area (computer science) or knowledge of pedagogical approaches to teaching computer science. Therefore, there is no motivation to ensure adequate preparation of the computer science teachers. In the colleges and universities in the United States there are no prerequisites for teaching that would expose faculty to teaching methodologies or field experience. Very few schools of education (if any) provide methods courses for prospective computer science teachers [55] because there is no national teacher certification in computer science. There is no blueprint that outlines a process for exposing teachers to techniques and tools that could help them become successful, effective, innovative computer science teachers.

Programming is an art that includes problem solving skills and effective strategies

for program design and implementation as well as the knowledge of programming tools and languages [5]. Although The Association of Computing Machinery (ACM) Educational Council is in the process of creating a new website, "Technology that Educators Hail (TECH)," that is intended to provide a central, organized collection of links to ready-for-prime-time technology resources [78], there are presently no such repositories of pedagogical tools that teachers can visit with the intention of finding a tool that is effective and appropriate for their teaching environment. There is presently no rating system or ranking of the tools available so that the teacher can distinguish between an effective and an ineffective tool and the reasons for the tool's rating.

Of course, tools are just learning/teaching aids. It is necessary to consider educational issues when deciding to use these technical aids. However good the materials are, they can still be used either effectively or ineffectively [5]. Teachers that choose to use pedagogical tools in their introductory programming courses do so for reasons. By investigating the perceptions of teachers using tools in introductory programming classes, this study identifies the reasons behind the adoption of the tools by the teachers.

3.3 Learning

Computer science education is not just about the teacher or the teaching. Computer science education is strongly based upon the higher tiers of Bloom's cognitive taxonomy, as it involves design creativity, problem solving, analyzing a variety of possible solutions to a problem, collaboration, and presentation skills [55]. Learning to program is generally considered hard
[18, 91, 94, 116, 149, 151, 177, 214, 225], and programming courses have generally experienced high dropout rates [15, 39, 47, 91, 118]. Learning to program is not a one-step process. It involves several activities, e.g., learning the language features,

program design, and program comprehension [5].

In recent years, computer science educators have argued that "the introductory course is so critical that students may be hopelessly lost if they do not have the right kind of experience from the start [213]". Engaging students is critical to deep learning [91] and today's learners are motivated in ways unlike any other generation. They are called the "Nintendo Generation," having grown up immersed in a world of technologies and computers [91, 158, 205]. They play video games; they listen to music on digital compact disks; they help their parents program the computerized controls of digital media players; their primary method of communication is via text-messaging and social networks. These experiences have given our students a different way of interacting with information compared with previous generations [190].

Traditional methods of teaching with the blackboard and a piece of chalk will not entice this Nintendo generation. Traditional methods of engagement have been replaced by active learning experiences where the introductory programming student uses a workbench to independently create and test objects [28], or creates stories using 3-dimensional worlds [6], or visualizes how a Java program is interpreted where method calls, variables, and operations are displayed on a screen as the animation goes on, allowing the student to follow the step by step the execution of a program [107].

For students to achieve better comprehension of programming and to enhance understanding of programming concepts, researchers have designed and developed a myriad of pedagogical tools. The variety and number of teaching technologies has grown tremendously over the past decade. Today, effective education involves more than simply well-structured formal lectures in the classroom. Research has shown that students learn best when actively engaged in the learning process [30, 42, 202].

This is enough of an impetus for every computer science teacher to create a learning environment that supports active engagement. Including the appropriate pedagogical tools as part of a teaching repertoire can facilitate active learning.

This study investigates the teacher's perception of the characteristics of pedagogical tools as they relate to student learning.

3.4 The Importance of Teacher Perceptions

Teachers are key agents of change and it is the teachers' beliefs and perceptions that influence their choices in the classroom [168, 169]. Manns investigated factors affecting the adoption and diffusion of innovations. Her research cites multiple sources dealing with the importance of perceptions in the adoption of an innovation [146]. "The [individuals'] perceptions of the attributes of innovations, not the attributes as classified by experts or change agents, affect its rate of adoption" [195]. Manns' work followed the work of Kishore, Iivari, Green, and Brancheau, who, when investigating the influences on innovation use, considered individuals' perceptions of the variables under investigation, rather than the attributes of the variables as potentially defined by others [146].

This study follows a similar approach by investigating the teachers' perceptions of the characteristics of pedagogical tools used in introductory programming classes rather than to investigate the individual characteristics in an empirical study. The results provided from this study add a dimension that has not yet been addressed fully by past research. As discussed in Chapters 4 and 5, past studies have presented analytic studies that examine a tool and its conformance to a certain set of criteria, anecdotal evaluations of specific tools, empirical studies that present quantitative or qualitative results about an individual characteristic of specific tools.

This research is different. Where other studies are limited to investigating one

specific tool or one category of tools, this study examines the characteristics that teachers perceive to contribute to the effectiveness of a tool, regardless of category. The results of this study provide information to present and future tool developers about the characteristics that are important to the programming teacher and how to present their tools to encourage a more efficient and more effective widespread adoption of the tool into the teacher's curriculum.

Past research on acceptance of technological innovations has confirmed that communication sources have the power to alter individual perceptions [146] and that perceptions of the attributes of innovations affect its rate of adoption [195]. The results of this study, when communicated to other computer science educators, provide information that may serve as a guide in the adoption of pedagogical tools in an introductory programming course.

3.5 Research Questions

Different techniques that have been implemented for evaluating pedagogical tools include anecdotal approaches, analytic techniques, and empirical evaluations [86, 100].

Many studies present anecdotal evaluation by having the tool developers discuss what effects they have observed when using this tool in their classes or what teachers and/or students have reported about the use of the tool. Such studies include work done by Sanders with Jeroo [199], work done with image processing libraries at Swarthmore College [224], and work done with Alice [117].

Analytical studies examine tools investigating a specific aspect or set of criteria of the tool with the goal of evaluation. Given an accepted set of criteria, evaluation is largely by direct inspection and inference [85]. The aim is to provide an assessment while avoiding the overhead of extensive empirical data collection. "Effectiveness, in

this case, boils down to usability–the fewer usability problems defined, the more effective the tool [100]." Gross describes an interpretive framework for evaluating novice programming environments and the use of this framework in evaluating the BlueJ IDE [85].

Empirical studies present quantitative and/or qualitative observational data and then analyze and transform the data into statements that make references about the tools [86, 100]. This type of study has been done with several categories of pedagogical tools answering questions such as "Is there any difference in learning when previous programing experience is taken into account? (Is the effect of the tool the same for novice and experienced students?) [184]" and "Does exposure to this tool increase the retention of students into the next course (CS2)? [160]."

Finally, there are summaries of the different techniques citing studies that exemplify each type [86]. But in most cases, the research involves one specific tool or one category of tools.

Studying individual pedagogical tools is no longer sufficient. The rate of technological advancement mandates a different, more comprehensive approach. More benefits result from determining the characteristics of *any* successful tool regardless of the tool or the category to which that tool belongs. In general, what characteristics contribute to the effectiveness of a successful tool in the eyes of the teachers?

There is significant research that is specific to teaching and learning programming [5, 20, 37, 39, 127, 131, 140, 187, 197, 214, 223, 225, 227]. Research done by Caspersen [39] supports the hypotheses that revealing the programming process to novices eases and promotes the learning of programming and that teaching skills as a supplement to knowledge promotes the learning of programming. His work focuses

on the foundations of learning theory for programming education, the indicators of success for learning and performance in introductory programming, and methods to educate novices in the skills of programming.

This study adapts Caspersen's hypothesis by investigating the perceptions of the teachers of introductory programming classes who use pedagogical tools to ease and promote the teaching and learning of programming. Visualizing an algorithm, watching an object's behavior in a microworld, and directly manipulating the state of an object are all ways of revealing the programming process with the use of the appropriate pedagogical tool.

This study focuses on the perceptions of teachers and proposes the following hypotheses based on the teachers' perceptions:

1. Using pedagogical tools in introductory programming classes eases and promotes the learning of programming.

2. Using pedagogical tools in introductory programming classes eases and promotes the teaching of programming.

3. Effective pedagogical tools used in introductory programming classes have common characteristics.

4. Tools that teachers consciously choose NOT to use in introductory programming classes have common characteristics.

5. Teachers initially choose to use a pedagogical tool because of a perceived task-technology "fit."

This study investigates the characteristics of tools that are perceived by teachers to contribute to the tool's effectiveness with a goal of determining the characteristics that are common to successful tools. As examples: Does Karel J. Robot (a

microworld) share characteristics with ObjectDraw (a library) that influence (or fail to influence) the tool's effectiveness? Do programming teachers choose to teach Alice (a microworld) for the same reasons they choose to use BlueJ (an IDE)? Does GameMaker (Game Making Software) share characteristics with Lego MindStorms (Robots) that ease and promote learning programming?

The following research questions shaped this study:

1. What influences the use of pedagogical tools in the introductory programming class? Specifically, what is the primary reason a teacher chooses to use a particular tool in an introductory programming class?

2. What are the perceived characteristics of an effective pedagogical tool used in an introductory programming course (effective is defined as: eases and promotes the teaching and/or learning of programming)?

3. What are the perceived characteristics of a pedagogical tool that hinder (or get in the way of) teaching and/or learning in an introductory programming course?

In an effort to address the hypotheses and to begin answering these research questions, this study examines the responses of teachers through an on-line survey administered between February 4, 2009 and April 2, 2009. Further information was collected from a subset of these teachers through phone interviews. Through these interviews, data pertaining to the underlying motivations and implications of using tools were collected. Insight into the motivation of tool developers in the design, development, and deployment of their tool was collected through in-person or phone interviews with several of the tool developers. Chapter 6 details the research methodology used in this study.

3.6 Limitations of the Study

This study does not represent a random sampling of teachers. The teachers participating in the survey satisfied one or more of the following criteria:

1. Participated as a reader for the Advanced Placement Computer Science (AP CS) Examination in June 2008 and expressed a willingness to participate in this study
2. Is a member of the AP CS Electronic Discussion Group (AP CS EDG)
3. Was an attendee at SIGCSE 2009
4. Was a participant in a teacher workshop or summer institute presented by this researcher between 2003 and 2008

The results of this study provide perceptions of a limited population of introductory programming teachers. The teachers surveyed are most likely at a higher level of professional involvement in professional conferences, professional development workshops and activities, and discussion groups focusing on computer science education. The responses of the sample were not taken to generalize to the population but rather to acquire in-depth information from those who are in a position to give it [19]. The teachers involved in this study are knowledgeable computer science educators actively involved in the computer science education community.

3.7 Significance of the Study

The significance of this study is the information that the results provide to tool developers about teacher perceptions of effective pedagogical tools and to teachers selecting and evaluating tools to use in their programming classes. Previous studies involve one specific tool or one category of tools. Studying individual pedagogical

tools is not sufficient. Technology is constantly changing. Good pedagogical tools come and go to fit with the technologies of the time. Existing tools will continue to be improved and new tools will continue to be developed.

Although the results from past studies provide meaningful information about a single tool, or a category of tools, more benefits will result from investigating the characteristics of any successful tools regardless of the tool or the category to which that tool belongs.

An effective pedagogical tool enhances and promotes teaching and/or learning. This study provides information on the characteristics that teachers perceive to contribute to the effectiveness of pedagogical tools used in introductory programming classes. The educators involved in this study are most likely at a high level of professional involvement and provide valuable information. Their perceptions are based on their experiences using these tools.

In addition, this study aims to shed light on the reasons that teachers choose to use (or not to use) tools in their classes and the reasons behind these decisions. Past research on acceptance of technological innovations has confirmed that communication sources have the power to alter individual perceptions [146] and that perceptions of the attributes of innovations affect its rate of adoption [195]. This study informs present and future tool developers about the teachers' perceptions and of how best to present their tools to encourage a more efficient and more effective widespread adoption of the tool.

The results of this study also provide information to teachers when deciding to integrate new tools into their curriculum. This research is based on the responses of teachers using pedagogical tools in introductory programming classes. These teachers are content experts and most likely at a higher level of professional

involvement in professional conferences, professional development workshops and activities, and discussion groups focusing on computer science education. The perceptions they offer are based on experience in the classroom.

The results of this study provide information about tool characteristics that are perceived to be important to these teaching experts. Communication of these results have the power to alter individual perceptions [146] and could serve as guidelines for teachers when choosing tools to incorporate into their programming courses thus affecting the rate of the tool adoption [195] and acceptance in the computer science education community.

Chapter 4

Teaching and Learning: An Overview

4.1 Introduction

Programming is a difficult skill to learn [18, 91, 94, 116, 149, 151, 177, 214, 225] yet learning to program is a primary goal in many introductory computer science courses. Leaders in the field of computer science education argue that the introductory course is so critical that students may be hopelessly lost if they do not have the right kind of experience from the start [213]. A negative experience can easily discourage novices from continuing in computer science and encourage them to pursue other fields of interest. This study investigates characteristics of pedagogical tools perceived by teachers to contribute to a positive first experience in the introductory programming class, making learning to program easier, more enjoyable, more understandable, and more appealing.

Introductory programming classes are smaller and more diverse than they were three decades ago. In addition to those students who are declared majors, computing educators attempt to attract and retain able students who have the ability to succeed in the field of computer science but are more likely to select other options. Sheila Tobias coins these students as "second tier" students [215]. Tobias defines "second tier" students as students not pursuing science in college for a variety of reasons [215]. These students are, by no means, second rate students or students incapable of learning science. Although Tobias generalizes her study to introductory *science* courses (physics, chemistry, mathematics), the situation is similar when we consider introductory *computer science* courses. Introductory computer science courses include many capable students that choose not to pursue computer science for a variety of reasons.

Many research studies on predicting success in introductory programming courses confirm that math and science ability contributes positively to success [26, 35, 110, 198, 227]. Many other studies investigate the relationships among learning styles and success in computer science [34, 69, 76, 214, 227]. Introductory programming classes include students with a variety of learning styles and a wide range of mathematical abilities. To accommodate different learning styles and a more diverse student population, all students, including these "second tier" students, should be welcomed into the computer science classroom, encouraged to participate in the activities, and offered a chance for success.

The educator's goal should be to demonstrate that computer science is not only for the student who thrives on math and science but can also be attractive to the student who is presently choosing other options. To help reach this goal, many programming teachers incorporate pedagogical tools into their curriculum with hopes of attracting and retaining students in their courses. Most of these pedagogical tools are designed with a goal of making learning to program easier, more enjoyable, more understandable, and more appealing. [135, 202].

This research study focuses on the characteristics of the tools that are common across categories and how these characteristics help ease and promote the teaching of and learning of programming. The results of this study are based on the perceptions of teachers, the most influential factor and the connecting link between the tools and the students in the class. The results identify the characteristics of the tools that address a population of students that includes the "second tier" students.

4.2 Learning Styles

People learn in different ways. There are many learning theories, categorizations of learning styles, and patterns of learning. The two learning style models that are

most commonly referred to in computer science education research are the Felder-Silverman Learning Style Model [8, 131, 214, 232] and Kolb's Learning Style Model [8, 34, 80, 120, 137, 179, 233].

The Felder-Silverman Learning Style Model classifies students according to four dimensions [214]:

- active learners (learn by trying things out, working with others) $\longleftrightarrow$ reflective learners (learn by thinking things through, working on their own)
- sensing learners (concrete, practical, oriented toward facts and procedures) $\longleftrightarrow$ intuitive learners (conceptual, innovative, oriented toward theories and meanings)
- visual learners (prefer pictures, diagrams, flow–charts) $\longleftrightarrow$ verbal learners (prefer written or spoken explanations)
- sequential learners (learn in incremental, orderly steps) $\longleftrightarrow$ global learners (holistic, learn in large leaps)
- inductive learners (prefer explanations that move from the specific to the general) $\longleftrightarrow$ deductive learners (prefer explanations that move from the general to the specific)

Felder notes that for at least the past decade, engineering instruction has been biased heavily towards the intuitive, verbal, deductive, reflective, sequential learning styles, yet few students fall into all of these categories [67]. Work done by Thomas with software engineering students confirms this [214]. Although a study conducted by Allert at the University of Minnesota Duluth did not find any significant relationship between performance and learning style in the sensing-intuitive dimension and the sequential-global dimension, the study did confirm that reflective

and verbal learners experienced more success in CS1 and CS2 courses than did those students classified as active or visual learning styles [8]. The learning style dimensions of this model are continua and not either/or categories. A student's preference on a given scale (e.g. for inductive or deductive presentation) may be strong, moderate, or almost nonexistent, may change with time, and may vary from one subject or learning environment to another [67].

The second common learning assessment tool used in science, engineering, and computer science is Kolb's Learning Style Model and is based on the premises that learning is a four stage cycle involving four adaptive learning modes [34]:

- concrete experience (learning through feeling)
- reflective observation (learning through watching and listening)
- abstract conceptualization (learning through thinking)
- active experimentation (learning through doing)

Kolb's learning behaviors are based on two dimensions: perception and processing. Along the perception dimension, there will be a preference between concrete experience (looking at things as they are without change) and abstract conceptualization (looking at things as concepts and ideas after internal processing). The processing dimension is a continuum between active experimentation (taking a conclusion and trying it out to prove that it works) and reflective observation (taking a conclusion and watching to see if it works). Each individual's learning behavior is based on these continua.

Kolb's learning theory views learning as a cyclic process and is based on the proposal that learning originates in concrete experience. Learning depends on experience but also requires reflection, developing abstractions, and active testing of

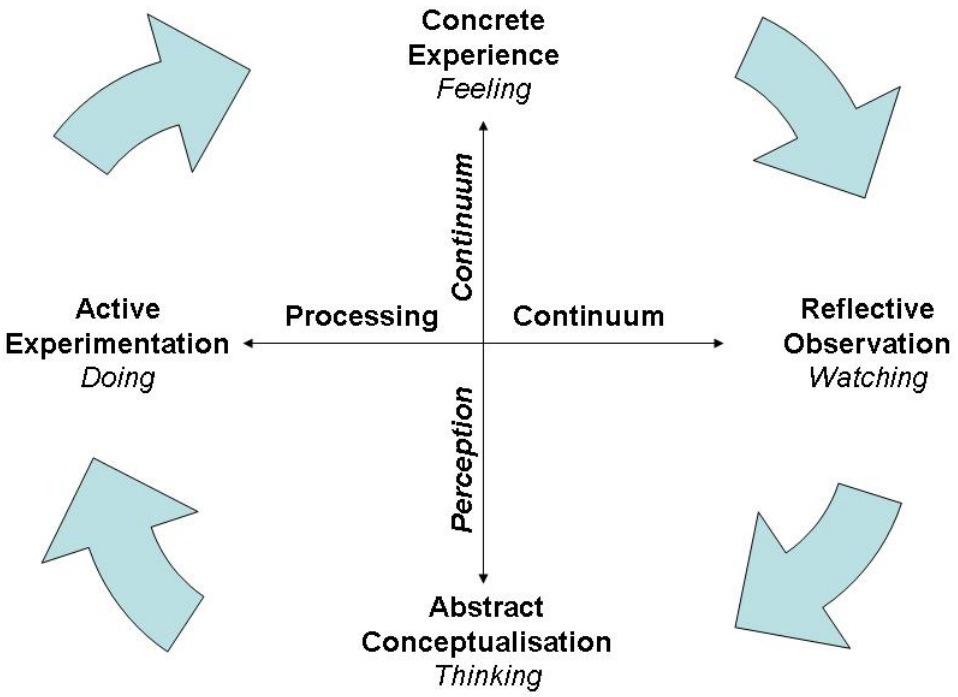

Figure 1: A Simplified Version of the Kolb's Learning Cycle [44]

these abstractions [233]. Learning to program involves several activities, e.g., learning the language features, program design, and program comprehension [5] and the programming student visits all four quadrants, depending on whether a student is trying to solve a problem, applying skills, or understanding and identifying the relationship between concepts [34]. Different learning styles come to play at different times during the programming process.

The learning cycle is usually entered starting with "Why" to motivate the material and also give Felders global learner the "big" picture. This motivation is linked to the concrete and reflective quadrant. Then the cycle progresses clockwise through the quadrants. "What" follows where definitions which are abstract and require reflection are presented. This kind of material appeals to the sensing learner in

Felders model. Then "How" demonstrates how the theory can be applied which involves active learning. Fedler's intuitive learners are more comfortable here than they were in the previous quadrant. Finally the cycle ends with "What if" which links to the next topic in the course [76]. This allows the student to synthesize and evaluate the current topic.

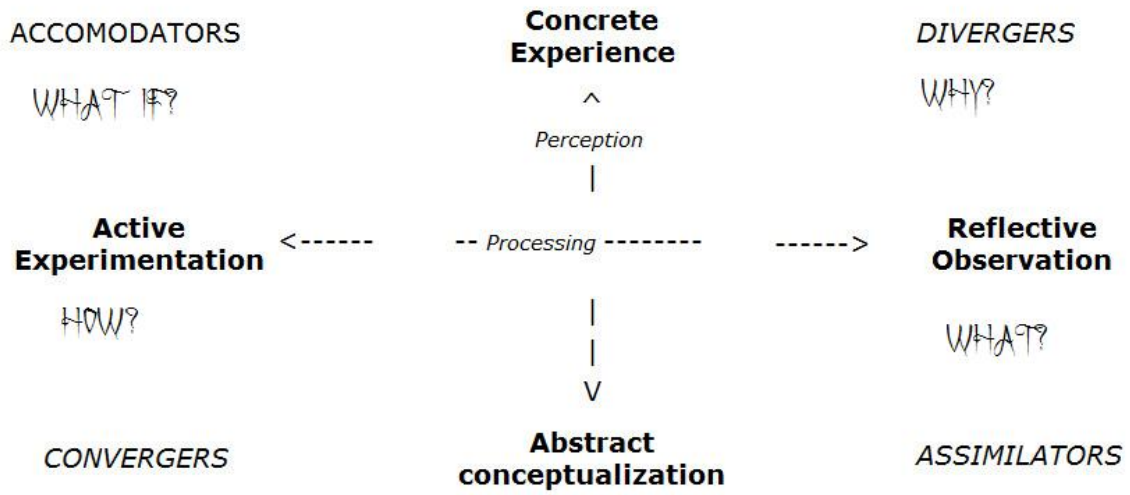

Figure 2: Based on Kolb's Learning Cycle with Identified Quadrants [119]

Kolb's Learning Style Inventory Technical Specifications states that educational experiences shape one's learning style by instilling positive attitudes toward specific sets of learning skills and by teaching students how to learn [120]. Although teachers can not force a particular learning style on a student, learning styles are influenced by experiences in the classroom. Early education teachers have the opportunity to guide the student to (or away from) a particular learning style, a preference to one quadrant over the others.

Introducing computing in the elementary grades provides an opportunity for educators to eliminate the "second tier" concept by addressing all learning styles and instilling positive attitudes toward the computing disciplines. The visual

environments that many computing pedagogical tools provide may contribute to addressing the different learning styles. This study investigates pedagogical tools as perceived by the teachers of introductory programming and the characteristics of these tools that ease and promote the learning of programming for students of all learning styles.

Kolb defines four learning styles [119], one for each quadrant created by the processing and perceiving axes (see Figure 2).

- *Divergers* take experiences and think deeply about them, diverging from a single experience to multiple possibilities. They like to ask "Why," starting from details to constructively work up to the big picture. They enjoy participating and working with others. They like to learn with hands-on exploration that leads to discovery. Students interested in Arts, English, History and Psychology are comfortable in this quadrant.

- *Assimilators* prefer to think than to act. They ask "What is there I can know?" and like organized and structured understanding. They prefer lectures for learning, with demonstrations where possible, and will respect the knowledge of experts. They will also learn through conversation that takes a logical and thoughtful approach. They often have a strong control need and prefer the clean and simple predictability of internal models to external messiness. The best way to teach an assimilator is with lectures that start from high-level concepts and work down to the detail. They like to stay serious. Students who enjoy Mathematics and Physical Science are comfortable in this quadrant.

- *Convergers* think about things and then try out their ideas to see if they work in practice. They like to ask "how" about a situation, understanding how things work in practice. They like facts and will seek to make things efficient

by making small and careful changes. They prefer to work by themselves, thinking carefully and acting independently. Student in Engineering and Medicine are comfortable in this quadrant.

- *Accommodators* have the most hands-on approach, with a strong preference for doing rather than thinking. They like to ask "what if?" and "why not?" to support their action-first approach. They do not like routine and will take creative risks to see what happens. They like to explore complexity by direct interaction and learn better by themselves than with other people. As might be expected, they like hands-on and practical learning rather than lectures. Students in Nursing, Education, and Communications are comfortable in this quadrant.

Although the results were not statistically significant, studies done by Byrne found that those with Converger style performed best overall in a first-year programming course. The strengths of Convergers are said to be in problem solving, decision making, deductive reasoning and defining problems. This style combines many of the attributes which are required for successful programmers [34].

Nourishing learning styles primarily takes place in the elementary grades but all educators should focus on respecting and addressing different learning styles in the classroom. Computing educators and researchers have devoted considerable energy to the development of pedagogical tools intended to ease student learning with hopes of increasing the likelihood of student success in computing [86]. They have proposed a number of strategies to ensure that all learning styles are addressed in the computer science learning environments. Incorporating the appropriate tool at the appropriate time for the appropriate educational outcome can help address these multiple learning styles in the hopes of easing and promoting the learning of programming.

This study investigates pedagogical tools and the teachers' perceptions on characteristics of these tools that may address different learning styles.

4.3 Learning Styles and Teaching Students

> *The beliefs, feelings, and assumptions of teachers are the air of a learning environment; they determine the quality of life within it.*
>
> Neil Postman and Charles Weingartner

The results of a study conducted by Pope and Scott [176] suggest that teachers' views on knowledge and theories of learning affect their practice in the classroom. Yuen conducted a study that explored the perspectives held by twelve Hong Kong computer science teachers about teaching computer programming and the underpinning pedagogical assumptions reflected in their perspectives [231]. The following were among the themes that emerged from Yuen's study:

- Two teaching methods were evident from data analysis.
  - Teacher-dominated method: Teachers direct students' learning through textbook and lecture. Teachers are serving immediate needs of dependent, authority-centered, linear thinking students [231]. The students being addressed by this teaching method are verbal-sequential (learning through written and spoken explanations incrementally) on the Felder-Silverman Model and are comfortable being Assimilators (prefer organized structured understanding).
  - Subject-centered method: Teachers provide more information and use a variety of presentation methods. The responsibility for learning is placed upon the student while the teacher provides opportunities for learning to take place. Teachers illustrate with examples, metaphors, and pictures. Some teachers use mathematical examples to promote students to correlate their thinking in computer programming [231]. Although

touching on visual learners with various types of examples, learning is still guided by the teacher and favors reflective over active learners. Using mathematical examples favors Assimilators and Convergers. Learners in these categories prefer to think, stay serious, and work alone.

- The following were among the factors affecting learning that were perceived by teachers.
  - Some students have no learning motivation. Lack of relevance of the subject to their daily life impeded students' motivation.
  - Science students were more capable of learning computer programming than art students.

Lister suggests that there is more to predicting success in programming than cognitive factors and that perhaps we should observe and interview students [139]. Many studies in computer science education have been based on student perspectives about their learning process [60, 92, 98, 126, 132, 145, 157, 224, 230]. In Tobias' study, a student interview reveals the following responses to why students who could succeed in science choose other options [215].

- There is a difference in values between a person in humanities and those of a scientist: in science, there is a preference for "how" questions over "why" questions.
- There is no sense of community within the class; there is a void of personal expression.
- Exams tend to ask for exhibition of skills acquired rather than conceptual understanding.

- When asked to display a knowledge of so many concepts at once, it is hard to get a hold of things.

Clearly, the experiences of this student confirm that the class was not catering to those students who are Divergers (who like to work in groups and ask "Why" questions) or Accomodators (who prefer hands-on activities and opportunities for taking creative risks). Active, intuitive, visual, and global learners were not the focus in this student's science class. In *Unlocking the Clubhouse: Women in Computing*, Margolis and Fisher investigate why girls do not enroll in computer science classes [147]. The following reasons are among those listed.

- Courses are taught in a dry, abstract style focused on language details rather than applications.
- The classroom climate in unfriendly to girls.
- Computing is a male activity.

Again we see that the comments reflect dissatisfaction with an environment that caters to sensing, sequential learners. The Grand Challenges paper [150] correctly nominates, as a subchallenge, the participation in research-based challenges whose purpose is to promote an improved image of "computing". It is not enough to improve our image. We need to improve our understanding of what is educationally interesting to those people outside the "clubhouse" [139] and we need to address their learning styles.

What makes learning take place? How does (or can) a teacher affect the student's learning? Chickering and Gamson offer seven principles based on research on good teaching and learning [42].

Good practice in undergraduate education :

1. Encourages contacts between students and faculty.
2. Develops reciprocity and cooperation among students.
3. Uses active learning techniques.
4. Gives prompt feedback.
5. Emphasizes time on task.
6. Communicates high expectations.
7. Respects diverse talents and ways of learning.

Good teaching and good learning take place in the classroom where the teacher is the most influential factor and the connecting link between the students and the pedagogical tools used in the classroom. This study investigates the teachers' adherence to these principles when choosing pedagogical tools for introductory programming classes.

The aspects of pedagogical tools that address different learning styles, support good practices in education, and ease the students' learning of programming are discussed in Chapter 5 as is the past research on the specific tools and categories of tools.

Chapter 5

Pedagogical Tools: Educational Research

> *The basic question is to decide what to do: either develop a tool for existing teaching and learning practices, or change teaching and learning practices by developing a new tool* [129].

A study by Palak and Walls sought to examine the relationship between teachers' beliefs and their instructional technology practices among technology-using teachers who worked at technology-rich schools to ultimately describe if change in practice toward a student-centered paradigm occurred. The results showed evidence that teachers' use of technology to support student-centered practice is rare even among those who work in technology-rich schools and hold student-centered beliefs. The study also indicated that these teachers continue to use technology in ways that support their already teacher-centered instructional practices [169]. Although this study focuses on K-12 teachers and the technology referred to was not necessarily pedagogical tools used in programming classes, these K-12 teachers are instilling attitudes about the use (or non-use) of technology in a classroom setting.

In referencing why teachers do not accept and actively use software tools in their programming classes, Levy cites two important findings grounded in literature [135]:

- There is an inconsistency between positive reactions of teachers towards innovations and the fact that those same teachers do not bring those innovations into their classrooms.
- Teachers find it difficult to relinquish their position of authority and they resist changes that might move the center of learning to the student, "reducing" the role of the teacher to that of a facilitator.

Becker and Riel address these concerns by summarizing the educational theorists Dewey, Piaget, and Vygotsky in providing the following teaching best practices [17]:

- Design activities around teacher and student interests rather than in response to an externally mandated curriculum.
- Have students engage in collaborative group projects where skills are taught and practiced in authentic contexts rather than in a sequence of textbook exercises.
- Focus instruction on students' understanding of complex ideas rather than on definitions and facts.
- Teach students to self-consciously assess their own understanding, in contrast to multiple choice testing.
- Model learning, rather than presenting oneself as fully knowledgeable.

A pedagogical software tool cannot stand on its own; rather, it must be integrated into the curriculum [135]. It's not the tool alone that supports student learning in this curriculum. It's how and when the teacher uses the tool.

Section 4.3 lists seven principles based on research on good teaching and learning offered by Chickering and Gamson [42]. This study investigates pedagogical tools used in introductory programming classes and the teachers' perceptions of the characteristics of the tools that contribute to its effectiveness. This study investigates the teachers' adherence to these seven principles of good teaching and learning when choosing pedagogical tools for introductory programming classes.

A discussion of literature pertaining to the individual tools and tool categories as they relate to introductory programming courses follows. For the purpose of this

research, the tools are classified into different categories. Included here is a brief explanation for each category and the past research related to the tool category or individual tools in the category.

5.1 Visualization Tools

Visualizations are graphical displays of information. Program visualizations consist of different graphical objects (often animated) and textual objects visualizing the execution of programs [184]. Visualization technology aims to help students understand how algorithms work in order to ease the problems of learning to program [126]. Examples of visualization tools referenced in this study include Jeliot 3, a program visualization application that visualizes how a Java program is interpreted [107] and jGRASP, a tool providing automatic generation of control structure diagrams for code visualization, UML class diagrams for architectural visualization, and viewers for dynamic views of primitives and objects [109]. Jeliot 3 and jGRASP offer dynamic visualization techniques. RAPTOR is an iconic programming environment, designed specifically to help students visualize classes and methods and limit syntactic complexity. RAPTOR programs are created visually using a combination of UML and flowcharts [38].

The purpose of any visualization tool is to improve the learning of the algorithms or concepts that are visually displayed. Many computer science educators may believe that visualization technology, under the right conditions, can greatly benefit learners and instructors alike but it is of little educational value unless it engages learners in an active learning activity [162]. Naps, et. al lists eleven commonly accepted best practices for Algorithm Visualizations (AV):

1. Provide resources to help learners interpret the graphical representation.

2. Adapt knowledge to the level of the user.

3. Provide multiple views.
4. Include performance information.
5. Include execution history.
6. Support flexible execution control.
7. Support learner-built visualizations.
8. Support custom input data sets.
9. Support dynamic questioning.
10. Support dynamic feedback.
11. Complement visualizations with explanations.

But "an educator must weigh carefully how to adapt and apply a visualization system since there is no single system or activity that is best for all learners [162]." In a study done by Parker and Mitchell [170], the visualization software used was intended only to supplement traditional teaching methods. In that study, animated demonstrations had a measurable impact on simple algorithms but complex algorithms provided no benefit. The authors also state that the student's learning preference has more to do with comprehension and retention than the effectiveness of an algorithm animation system.

There are mixed reviews of the effectiveness of visualizations used in programming classes [12, 100, 142, 174]. Only about half of experimental studies conducted on algorithm visualizations demonstrate a significant effect for the AV, and lack of significant improvement is often attributed to the AV not actively engaging the student [201]. Some say they do not significantly improve the learning [100] and others claim they help motivate students and aid in the understanding of how

algorithms work [114, 136, 184]. Research has shown that students learn best when actively engaged in the learning process [30, 42, 202]. Active learning involves students participating in some way that results with them thinking about what they are doing rather than just listening [202]. The more actively learners were involved in activities involving AV technology, the better they performed [100]. Naps, et.al identifies a taxonomy of engagement levels that are summarized by Urquiza-Fuentes and Liu [142, 218]:

1. *No Viewing.* This level refers to instruction without any form of accompanying algorithm visualization.

2. *Viewing.* This level can be considered the core form of engagement. A learner can view an animation passively, but can also exercise control over the direction and pace of the animation, use different windows (each presenting a different view), or use accompanying textual or aural explanations.

   The remaining four categories all include viewing.

3. *Responding.* The key activity in this category is answering questions concerning the visualization presented by the system. In the responding form of engagement, the learner uses the visualization as a resource for answering questions.

4. *Changing.* This level entails modifying the visualization. The most common example of such modification is allowing the learner to change the input of the algorithm under study in order to explore the algorithms behavior in different cases.

5. *Constructing.* In this form of engagement, learners construct their own visualizations of the algorithms under study. It is important to note that the

constructing form of engagement does not necessarily entail coding the algorithm.

6. *Presenting.* It entails presenting a visualization to an audience for feedback and discussion.

The results of Naps' study confirmed that learning improves as the level of student engagement with AV increase [162]. Traversing through the engagement levels will ultimately meet the needs of the Divergers, who prefer to start from details and work up to the big picture through hands-on activities, and the Accomodators, who like to explore complexity by direct interaction, and perhaps motivate "second tier" students to pursue computer science.

5.2 Microworlds

A microworld is a learner-centered world that is explored by directly manipulating objects in the world with a limited set of simple commands [86]. More generally described by Latour, "A microworld is a tiny world inside which a student can explore alternatives, test hypotheses, and discover facts that are true about that world [128]." Microworlds provide metaphors where storytelling can occur. Examples of microworlds referenced in this study include Alice [6], Greenfoot [82], GridWorld [83], Jeroo [108], and Karel [24].

Many microworlds are designed to provide a gentle introduction to object-oriented programming, provide concrete experiences with objects, and support visualizing objects in a meaningful context [23, 50, 60, 123]. The use of a programming microworld is based on a physical metaphor and focuses the students' on solving interesting problems. The use of microwords contributes greatly to decreasing the "distance" between the mental models or descriptions of algorithms in a natural language and their description in a programming language [230].

Henriksen and Kölling state that any tool to support an object-early approach for beginners must attempt to support the practical stages of Kolb's circle (experimentation, experience, observation) explicitly at the level of the fundamental concept: objects. Students must be able to manipulate, experiment with, and observe objects, not merely lines of source code [96].

Microworlds are pedagogical tools that can motivate students, enable creative self-expression, reduce technical barriers, and increase confidence and retention [60, 96, 117, 177]. These design goals are generally supported by the perceptions of teachers and tool developers teaching with microworlds and by students' reactions to programming with microworlds as recorded in interviews or on surveys in the documentation of past research studies:

- The students found both the programming environment (objectKarel) and the exercises interesting [230].
- Alice did seem successful at increasing our weaker students self confidence in their programming abilities [178].
- Students in non-majors introductory programming courses responded favorably to their experience of programming with Alice [98].
- In our experience the impact of Alice on student perceptions of and attitudes toward programming seemed largely, although not entirely, positive. In particular, we found the ability of Alice to generate laughter remarkable [178].
- The attrition of our most at-risk majors has been significantly reduced [50].
- Students found Jeroo to be a valuable tool and felt it serves as a good introduction to Java [60].
- Students appear to have more confidence in their own ability [200].

- Students have developed a more mature style of programming than in the past[60].
- Students have a better grasp of control structures, methods, and objects following the four-week introduction to programming using Jeroo [60].
- Most of the students believed that they had a stronger understanding of classes, objects, and how to use Greenfoot. They also believed that they had a higher comfort level with programming in general and technology in general [4].
- The course (Karel) is fun for both students and instructors; students understand the fundamental concepts early, allowing them to approach advanced topics with confidence [16].

Formal empirical research with microworlds has provided the following information

- A study by Mullins reports that retention data shows that the incorporation of Alice into the programming sequence has increased the number of students that pass the courses and decreased the number of withdrawals [161].
- A study conducted by Moskal, et. al supports the effectiveness of the Alice course for improving students performance in CS1, retention within computer science, and attitudes towards computer science. At-risk students that participated in Alice received significantly higher grades than at-risk students that did not participate in Alice [160].

There is not a great deal of formal research (quantitative or qualitative) on the use or effectiveness of of microworlds in introductory programming classes. Many fine computer science educators and researchers have offered suggestions to ease the learning of programming [95] or to encourage the use of microworlds to ease the

teaching of programming, but these are the suggestions of the educators. Empirical data based on learning behaviors and/or teaching pedagogy related to the use of microworlds is lacking.

5.3 Robots

In the context of this study, robots refer to the LEGO Mindstorms^{TM}system. This system extends the traditional Lego bricks with a central control unit (the RCX) as well as motors and various kinds of sensors [121]. The robots can be programmed to explore an environment, detect obstacles and lights, and solve simple problems [94]. Another example cited in this study is Robotran, a multibody modeling at UCL-CEREM which is entirely based on Matlab/Simulink [193].

As with the other pedagogical tools discussed, the most important goal in introducing robots into the computer science curriculum is to improve student learning [65]. Past research on using robots in introductory programming classes has mixed results. A study conducted by Fagin and Merkle in a core computing course required by all first-year students found that test scores were lower in the robotics sections than in the non-robotics ones, and the use of robots did not have any measurable effect on students choice of discipline [65, 66]. The results of a study conducted by Hasker suggest that students using robots in an introductory programming class learn as much as those engaging in traditional programming problems and that the students found programming the robots more interesting [94]. Studies by Davis et al. show that teaching with robotics does not necessarily hinder students from learning the basic programming skills in introductory classes [57]. Student surveys in a study conducted by Meyer resulted in 59% of the students felt that working with robots helped them to better understand algorithms, 64% indicated that working with robots helped them to better understand programming, and 71% said that more robotics material should be included in the course [155].

Summet, et al. personalized a Computer Science 1 course (CS1) by developing a curriculum that uses a robotics context to teach introductory programming in which a robot is provided for each student. Their results show a higher success rate in the robotics sections of CS1 than the non-robotics CS1 sections [209].

Because of the physical nature of a robot, there is an unavoidable hands-on approach to teaching. Robots encourage student experimentation at many different levels [130]. This approach is appealing to the Felder's active learners and to the Accomodators in Kolb's Learning Style Model.

5.4 IDEs

An integrated development environment is a software application that provides several utilities for programmers and software developers packaged into one bundle. This bundle usually includes a source code editor, a compiler or interpreter, automated building tools, and a debugger. Most IDEs provide a way to assist students in writing correct syntax in a language in which they may not be proficient, provide a simple interface to the language compiler, flag any syntax errors, and include a mechanism for running a program [188]. Some IDEs were specifically designed for introductory programming classes providing a simple, unintimidating environment with an interactive interface [7, 28, 188]. Examples of IDEs referenced in this study are BlueJ [28], DrJava [61], DrScheme [62], Eclipse [64], Greenfoot [82], JCreator [106], jGRASP [109], and Netbeans [163].
Some IDEs like BlueJ, DrJava, and jGRASP are designed specifically for introductory programming classes. Because of this gentle approach, these IDEs may fail to expose students to "real world" programming environments. IDEs such as Eclipse and NetBeans are targeted at professional developers and include a host of advanced features but may have learning curves are too steep for novice programmers.

"In an introductory class, a programming interface must be intuitive and the IDE should provide simple mechanisms for working around complications in the Java language that are pedagogic distractions (i.e. `public static void main(String[] args)`) [189]." The IDE for novice programmers should create an unintimidating atmosphere where the student is not overwhelmed with the features of the tool but instead the tool eases and promotes the learning of programming. BlueJ, DrJava, and jGRASP are IDEs that are designed specifically for the novice learning environment. [7, 53, 124].

BlueJ was developed for the teaching and learning of object-oriented programming [124]. The developers of BlueJ believe that Integrated Development Environments (IDEs) for object-oriented language should encourage users to develop and test individual classes rather than requiring users to always create complete programs [116]. They believe it is essential that the program development environment is unified with the programming language paradigm [196]. BlueJ is widely used in introductory programming classes at both the college and high school levels. This is confirmed by hundreds of list posts on the AP CS EDG [11] referencing the use of BlueJ and the numerous publications in computer science education that acknowledge BlueJ as the environment in which the research is based [13, 68, 71, 93, 122, 144, 171, 172, 182, 194, 221, 229]. Although this research does not directly prove or directly support the effectiveness of this IDE on student learning, it does confirm the widespread use of BlueJ within the computer science education community. There are, however, non-believers. The results of Shanmugasundaram, et.al indicate that BlueJ is not significantly more effective ($p > 0.05$) than Textpad in the learning of Java programming [203]. In their study, both sections were taught and tested with the same materials. These researchers observed that, for some reason, the students do not tend to stay with testing their classes using the object bench of BlueJ after using it in the early stages of learning.

BlueJ allows teachers to teach introductory courses differently than can be done without it. The tool allows for the avoidance of the dreaded `public static void main(String[] args)`. But teaching with BlueJ is not just about the tool. If the teacher does not

understand the pedagogical advantages of this environment and does not teach differently with the tool than without it, the learning benefits emphasized by the developers are lost.

DrJava is a lightweight development environment designed to evaluate expressions, instantiate objects and call methods in a similar way as an interpreted language such as Python. DrJava provides a good example of providing an intuitive interface that has the ability to interactively evaluate Java code [61, 175]. Students can enter and execute code one line at a time thus getting immediate feedback. Most of the literature written on DrJava focuses on the tool capabilities and not on the learning that results from using the tool or how the tool affects teaching in an introductory programming class.

The strength of both BlueJ and DrJava is active experimentation with Java constructs. Beginners do not have to write complete programs to experiment with simple expressions and statements, and get immediate feedback [167]. Olan presents a comparison of the pedagogical approaches used by BlueJ and DrJava as a guideline for teachers to select the IDE best suited to the teaching style used in the introductory course [167].

jGRASP is a lightweight IDE that provides automatic generation of visualizations that directly support the teaching of major concepts in CS1 and CS2 [52]. It provides both static and dynamic visualization capabilities [175]. Like BlueJ and DrJava, jGRASP is developed to ease the task of teaching and learning Java. jGRASP includes a workbench, an interactions pane, and also creates control structure diagrams to help in program understanding. The developers believe that each of the automatically generated software visualizations (CSD, UML, and Object Views) can be used to make learning to program a more enjoyable experience [53]. Most of the literature written on jGRASP focuses on the tool capabilities and not on the learning that results from using the tool or how the tool affects teaching in an introductory programming class.

JCreator seems to be a popular Java IDE in secondary schools. More than one-third of the posts from the AP CS EDG about IDEs reference JCreator [11]. One disadvantage is that Jcreator is only available for the Windows Operating System and many secondary

schools have adopted the Mac OS [11]. Very little (if any) formal research has been published on the use of JCreator in introductory programming classes. The Web site for this IDE claims that it is the perfect tool for programmers of every level, from the novice programmer to the Java-specialist [106].

Professional IDEs are designed to help software developers write programs more quickly and produce better quality code. This is not necessarily the focus in an introductory programming course. The advantages of a professional IDE may be outweighed by its complexity, requiring students to spend excessive time learning the tool.[175]. Chen and Marx [41] concluded that the complexity of Eclipse was manageable only if the instructor continuously demonstrated it in the classroom and asked the students to do the same. More importantly, the instructor needed to introduce Eclipse features gradually. Their observations also indicate that the Eclipse IDE is challenging to students but the challenge actually inspires the students' "can-do" attitudes. They also observed that after using Eclipse, most of their students showed a sense of confidence and were eager to share their experience with others. Deugo claims that students need to know how to use Eclipse to be competitive in todays corporate software market [59]. He based this on the results of a survey conducted in November 2006 by BZ Media. The survey reported that 66.3% of SD Times subscribers surveyed reported that developers within their organizations used Eclipse.

NetBeans is a free, open-source Integrated Development Environment for software developers [163] bought by Sun Microsystems in 1999. Although NetBeans is used in educational institutions, it is primarily used at the university level in advanced computer science courses (database management [36], applications programming [217], and software engineering [101]) and not in introductory programming classes.

To attempt to achieve the best of both worlds (a pedagogically sound learning environment and exposure to a "real-world" IDE), the pedagogical IDE developers are working with the professional IDE teams to create plug-ins for the professional environments.

With the DrJava plug-in, Eclipse is transformed from an intimidating professional environment to a pedagogic IDE with a reasonably simple interface [189]. The plug-in is intended to provide several features from DrJava to Eclipse, including an interactions pane, a simplified user interface, and a debugger integrated with the interactions pane. In addition, the DrJava plug-in can be combined with other pedagogic Eclipse plug-ins, enabling an instructor to create an environment with the specific capabilities required for a particular course [189].

The BlueJ and NetBeans teams are collaborating to create a tool (NetBeans IDE BlueJ Plugin) that offers a seamless migration path for students that supports the switch from educational tools into a full-featured, professional IDE [163]. The plugin enables NetBeans to work with BlueJ projects so that students can start to make use of NetBeans more advanced features without sacrificing the features and familiarity of BlueJ [28].

5.5 Games

This study references the Game Maker development software available through YoYo Games. The software involves drag-and-drop actions to program, has a very short learning curve, and does not require prior programming experience [185]. The games include backgrounds, animated graphics, music, and sound effects [77].

Many computer science educators recognize that students show enthusiasm for activities involving elements of games and so game development is often used in introductory programming classes as a motivational technique [45, 133, 180]. Games are used as examples and assignments in computer science courses [14, 123] and it is not uncommon to use games or game development to engage students early in the computer science curriculum with hopes of improving recruitment and retention [125, 159, 180].

Giguette incorporated a *pre-gaming* activity where students played a game-like version of each assignment before designing and coding their own programs [79]. Other studies have had students rewrite classic computer, arcade, or board games or to redesign these classic

games [99, 143, 211]. A study by Wescott investigated whether or not digital game *playing* improves the effective transfer of the students problem solving, critical thinking, logical, and programming knowledge from game playing to a formal programming environment [222]. For this study, the researcher discussed programming concepts and provided a mapping of concepts to the game being played. The students then used Scratch to play the game and to modify the code and as a final step, the student transitioned from Scratch to formal programming involving the C++ programming language. This research found that playing digital games improves overall programming outcomes.

Chamillard describes conducting a longterm study of the effectiveness of integrating computer games into the computer science curriculum using such tools as Game Maker, The Games Factory, Pie 3D Game creation system. The results of the study indicate that most students "do very well on the game assignments using the game development tools [40]."

Work done by Guimaraes and Murray used the gamemaking software, Game Maker. Game Maker provides and environment that is described as being intuitive and easy to use and that introduces students to programming through an objects-first approach. Through the analysis of the use of Game Maker during camps offered to university students and junior and senior high school students, several strategies that contributed to student engagement and learning were identified [87]. These included:

- It is beneficial for students to play an example game, modify that game and then create their own game implementing the features exemplified in the example game. (This addresses active, visual learners and the Accomodators who prefer hands-on and practical learning.)
- It is important to properly set expectations for what students can achieve in such a short time. (This addresses the sequential learners who learn in orderly steps and the Assimilators who like organization and structure.)
- It is important to provide time for students to work independently allowing them to

experiment on their own and develop their own individual learning processes. (This addresses the Convergers who prefer to work on their own and also to the Accomodators who like to explore complexity by direct interaction.)

- Requiring student presentations of their work provides motivation for students to achieve at higher levels.

Game Maker provides the opportunity for students to engage and experiment with basic computer science concepts in a motivating environment that provides immediate feedback [87]. The nature of games permits active engagement and increased enthusiasm levels. Learners can explore and experiment [183]. Using Game Maker and following the strategies above in an introductory programming class, can help to address the different learning styles. Designing and building a game can be a very useful way towards a more thorough understanding of the algorithms, data structures and other topics in computer science [183].

With the inclusion of games in an introductory programming curriculum, the hope is to motivate students to enroll in the course, to encourage success regardless of the student's preferred learning style, and to retain these students in the computer science discipline.

5.6 Libraries

In the context of this study, libraries refer to collections of Java tools that are designed to make programming easier for novices. These collections provide simplified features (e.g. GUI components) or provide rich collections of classes that are easily used by novices [32]. Libraries referenced in this study include ObjectDraw [166], the ACM Java Task Force Library [104], Java Power Tools [103], and the Media Computation Library [152].

Because teaching Java in an objects-first approach is often very difficult, computer science researchers and educators have developed libraries to ease that task [104, 103, 152, 166]. These libraries (toolkits) are developed to emphasize programming concepts from the beginning, requiring students to think from the start about the programming process with

a focus on methods and objects [31]. Libraries, in the context of this study, provide simplified features (e.g. GUI components) or provide rich collections of classes that are easily used by novices allowing them to build programs with impressive graphics. These toolkits are essential for pedagogy since introductory programming classes require that students understand and learn to practice algorithmics, encapsulation, and interaction [186].

The toolkits or libraries commonly used in introductory programming classes and referenced in this study include ObjectDraw [166], ACM Java Task Force (JTF) Library [104], Java Power Tools (JPT) [103], and the Media Computation Library [152].

The Java Task Force Library, though the youngest of those identified above, is described first primarily because an itemized list of development goals is included in the documentation of this tool. The principles adopted by the Java Task Force in the development of the JTF Graphics Library include [104]:

- Design for an object-oriented approach allowing for an "objects-first" pedagogy.
- Adopt a minimalist strategy. To avoid the proliferation of complex tools that do little to address the scale problems of Java, the Task Force has sought to minimize both the number and conceptual complexity of our packages. The concern of the Task Force is limited to those areas that currently constitute stumbling blocks to the effective use of Java.
- Promote flexibility for adopters. Introductory courses in computer science vary widely in philosophy, topic coverage, and approach. The goal of the library development is to empower teachers and students by providing tools to extend their reach. The library tools are intentionally general enough to support a wide variety of programming styles.
- Maintain conformance with the Java standard. Throughout our design, the Task Force sought to use the standard Application Programmer Interfaces (APIs)

provided by Sun Microsystems. The only occasions in which we have proposed alternative APIs are those for which there was clear evidence that the existing facilities were not working well at the introductory level. We have also made the ACM packages separable to make sure that no one is required to adopt parts of the nonstandard packages that they regard as unnecessary.

- Retain compatibility with earlier releases of Java.
- Support multiple environments.

Observations made by Mertz confirmed that students have benefited from the simplification of the Java language made possible by the ACM Java libraries.
The simplicity of graphics programming using the JTF Library keeps students engaged by providing a tool for the teaching and learning of basic concepts in an interesting way [154]. The work of the JTF committee resulted in a collection of Java-based resources which simplifies the teaching and learning of computer science [153].

In designing the ObjectDraw Library the developers were very conscious of the fact that the advantages of using a library are offset by the disadvantage of hiding portions of "real Java" from students [166]. As with the JTF Library, the scope of the ObjectDraw library is carefully limited. As with the JTF Library, in using the ObjectDraw Library graphics classes, the design deliberately mimics the methods provided by the standard Java Graphics class [166].

Bruce, Danyluk, and Murtagh list the goals of an introductory programming course taught using ObjectDraw at Williams College [33]:

- Use an object-first approach, requiring students to think from the start about the programming process with a focus on methods and objects. (This approach addresses the Assimulators who prefer to start with high-level concepts and work down to details.)
- Use graphics and animation extensively. Experience in an earlier version of this

course convinced us that graphics can be an important tool both because students are able to create more interesting programs, and because graphic displays provide students with visual feedback when they make programming errors. (This addresses the needs of visual learners.)

- Introduce event-driven programming early. Most programs students use today are highly interactive. Writing programs that are similar to those they use is both more interesting and more "real" to the students. (This addresses the needs of sensing learners and Accomodators who like hands-on and practical learning.)

The ObjectDraw Library supports this "OO-from-the-beginning" approach [33]. It is evident that it is not the tool alone that is effective but how the tool is used within the introductory programming course.

Java Power Tools (JPT) is a Java toolkit designed to enable students to rapidly develop graphical user interfaces in freshman computer science programming projects [186]. Rasala, et al. note that one pedagogical problem in teaching OOP is that the small-scale textbook examples are not of a size that convinces a student that OOP is worth all of the fuss and until a student experiences a large project in which the value of OOP is evident there is skepticism that the complications of defining classes and methods are valuable [186]. As with the other libraries mentioned, the JPT Library provides rich collections of classes that are developed to be used by novices. Students entering the field of computer science now come with the expectation of learning to build programs with impressive graphics and flexible user interfaces [181]. Libraries are tools that make this possible in an introductory programming class. The potential for using Java to build quality user interfaces in introductory courses will be limited unless students can implement such interfaces easily [181]. The philosophy of the developers of the JPT toolkit is to provide students with the foundation necessary for building quality programs and then to let them use their talents to solve interesting problems that illustrate the breadth of computer science applications [181]. This philosophy addresses the needs of the Accomodators, who take creative risks and explore complexity by direct interaction, and the active learner.

Media Computation (MediaComp), developed at Georgia Institute of Technology (Georgia Tech), is a collection of cross-platform libraries to manipulate pixels in a picture and samples in a sound [90]. The purpose of these libraries is to allow for the development of exciting programming applications to be accessible by students in an introductory programming class, many of whom Tobias would classify as "second tier" students (capable but choosing fields other than computer science). The media computation course taught at Georgia Tech has been majority female, and women succeed at the same or better rates than the male students [90]. Similar improvements in success rates in media computation courses have been seen among underrepresented groups by Sloan and Troy at the University of Illinois at Chicago [204].

From experience using the Python version of MediaComp in introductory programming classes (CS 0.5) , Sloan and Troy believe the multimedia approach is a good one because most students enjoy multimedia already and are thus likely to be very interested in manipulating images, animations, and music themselves [204]. Sloan and Troy use the Georgia Tech approach of allowing a student to first work with the multi-media material itself, then with programming shortly thereafter [204]. Their results from working with Media Computation in this CS 0.5 course show a very low WFD rate (withdrawal, earn an F, or earn a D). The students in this course report that the course is highly relevant. The results of Sloan and Troy are from a pool of students that was over 25% African-American and Hispanic [204].

The theme of Georgia Tech's Computer Science 1 Course using the MediaComp library is: Computation for Communications [88]. Judging by survey responses of students exposed to the MediaComp Library, Forte and Guzdial conclude: Students seem to think that media computation as a subject is relevant and interesting and a number of students commented on the fact that they feel it is important to understand how the programs they use work [74].

An "Introduction to Media Computation" course that is specifically designed for

non-majors at Georgia Tech incorporates the following elements [192]:

- Creativity: to counter the reputation of CS as boring. (Supports Accomodators.)
- Relevance: to illustrate that CS concepts are applicable to non-majors. (Supports sensing learners.)
- Collaboration: to counteract the stereotypes of computing as asocial and inhospitable. (Supports Divergers.)
- Restricted registration (non-majors): to reduce intimidation and support a hospitable culture.

The course addresses the myths that computer science is boring and asocial, that the material is irrevelant to non-majors, and that the field of computer science is an intimidating, uninviting culture. Media and computation together provide a motivating framework for many students, and encourage some students to excel who would otherwise prefer not to try [74]. This approach addresses Tobias' "second tier" population. But it is not the MediaComp library alone that provides this framework. It is how this library is used in the learning environment.

5.7 Summary

Gross and Powers evaluated assessments on selected novice programming environments. They noted that most of the assessments were done by the developers of the environments and that many assessments conducted were more opportunity-directed than problem-directed. Tool developers are experts at using their environments, and have often developed an associated pedagogy designed to focus on the strengths and uniqueness of their environments [85]. The tools may not be effective in and of themselves but when the pedagogies are offered by the tool developers, frameworks for the proper use of the tool in a learning environment are provided. Gross and Powers note that some of the least studied questions are those that focus on how the environments impact a student's learning process and understanding from a formative perspective. Although some tools

clearly attempt to address different learning styles, there is little formal research that supports this, or even considers this as relevant.

The pedagogical tools that are used in introductory programming have changed what computer science educators teach and how they teach programming. Teaching an introductory programming course can be challenging, especially if the course enrollment includes students with divergent goals and different learning styles. By engaging students with hands-on learning activities, students come to view computer science as a fun, interesting, and a broadly applicable discipline. An initial positive experience with computer science can encourage students to continue studies within the discipline [10]. If computer science is not perceived as interesting or useful, students fail or drop out [74].

This chapter has discussed specific categories of pedagogical tools and the research done in each of the categories as it relates to the learning of and teaching of programming. Although some studies are based on empirical data and defined research methodologies, much of the information collected on the use of pedagogical tools focuses on student perceptions and teacher observations in the learning environment. Many of the studies give results that are based only on the developer's perspective.

Most past research in the area of pedagogical tools investigates one specific tool and its effectiveness (proven or observed) or one category of tools (e.g. Visualization tools) and the effectiveness of this type of tool as related to student learning.

This research study is different. It investigates the effectiveness of tools across categories. Good pedagogical tools come and go to fit with the technologies of the time. Existing tools will continue to be improved and new tools will continue to be developed. This study investigates the characteristics that are perceived to contribute to the effectiveness of pedagogical tools regardless of the tool category. It serves as a a resource for teachers, present and future tool developers, and those involved with computer science curriculum development.

The results of this study are based on the perceptions of teachers using pedagogical tools in the introductory programming course. Information gained from perceptions is important and can be a starting point for more formalized research. "The [individuals'] perceptions of the attributes of innovations, not the attributes as classified by experts or change agents, affect its rate of adoption [195]." Perceptions that teachers have about pedagogical tools used in introductory programming classes and the communication of these perceptions can affect the rate of adoption of the tool within the computer science education community.

Chapter 6

Research Methodology

6.1 Introduction

Mirriam-Webster defines *success* to be a favorable or desired outcome [156].

The success of a pedagogical tool designed for introductory programming courses depends on the tool's effectiveness and the dissemination of information about the tool to the population of programming teachers. Programming is an essential skill for students wishing to pursue the field of computer science [118, 150] and the task of specializing programming environments for novices begins with the recognition that programming is is not an easy skill to learn [70].

The Conference on Grand Challenges in Computing [150] was held in Newcastle, Great Britain, on 30-31 March 2004 to identify problems in computing, where 'computing' was used in a generic sense to include computer science, software engineering, information systems, artificial intelligence, computer hardware, communications, Internet computing, and even aspects of cognitive science. The Grand Challenges strand on the educational aspects of computing presented an opportunity to identify and articulate the priorities and the challenges that face educators in computing. As defined for this conference, an education-based grand challenge must be such that, when achieved, it will:

- lead to substantial improvement in some significant aspect of the educational processes in computing;
- arouse curiosity and generate enthusiasm within the computing community;
- be international in scope and so have wide and significant relevance;
- be comprehensible and capture the imagination;
- have the capacity to bring about changes in attitude, changes in expectation, even

change at a social level [150].

One of the seven challenge areas in computing education addressed at this conference was *programming issues*. The objectives in this area were described as being straightforward. Educators would agree that it is desirable to employ six *right* attributes. One of these attributes was to teach in the *right* environment with the *right* support tools [150].

There is little discussion of and research on the teaching of programming as it relates to pedagogy. There is even less educational research that address how the process of learning affects or should affect instruction [39]. The pedagogy employed in the introductory programming class needs to accommodate different programming paradigms, different learning styles, and a more diverse student population. Chapter 4 of this study discusses teaching pedagogies, learning styles, and the research done in these areas as it relates to introductory programming classes. In an attempt to make programming more accessible, a variety of pedagogical tools have been developed. The literature review in Chapter 5 describes the research involving the individual tools and the various categories of tools used in introductory programming courses.

The purpose of this study is to enumerate and examine a list of characteristics that are perceived to contribute to the effectiveness of a pedagogical tool used in introductory programming courses, to determine those characteristics that are common to successful pedagogical tools, and to determine the teachers' motivations for choosing a tool to integrate into their curriculum.

6.2 Research Process Design

An *action* research methodology is defined by educational researcher S. M. Corey, one of the founding fathers of action research, as a process used so that practitioners can evaluate, improve, and steer decision-making and practice [46].

This dissertation follows a modified definition of action research methodology. The results obtained by examining the perceptions of teachers are intended to steer tool developers in

the development of new tools and in the improvement of existing tools and to guide teachers in the adoption of pedagogical tools in an introductory programming course. As past research confirms, perceptions of the attributes of innovations affect its rate of adoption [195].

This study employs a mixed methods approach primarily implementing a sequential mixed method design [212]. The study starts with a quantitative approach (survey) and proceeds with a collection of experiences from individual teachers (interviews). The quantitative component provides information on the tools most often chosen to be incorporated in an introductory programming course and the characteristics that are perceived to contribute to the effectiveness of the tool. While the quantitative portion provides greater breadth, the qualitative portion provides more depth to the results. In parallel to this, interviews with tool makers were conducted to provide insights from the developer's point of view, adding a parallel qualitative strand in this study. Together, the three strands yield more complete and meaningful results. Figure 3 uses five distinct design stages over three distinct strands to illustrate the mixed methods approach used.

- The *Conceptualization Stage* includes the design of the survey and the development of the interview questions.
- The *Experimental Stage (Methodological)* includes the data collection.
- The *Experimental Stage (Analytical)* includes analysis of the data.
- The *Inferential Stage* includes emerging explanations and inferences.

This research design and methodology are used in collecting information from both educators and toolmakers in order to understand their practices and perceptions involving pedagogical tools in introductory programming classes. The data collected provide answers to the research questions posed in Chapter 3, provide valuable information to tool developers, and provide a reference for teachers choosing to use pedagogical tools in introductory programming courses.

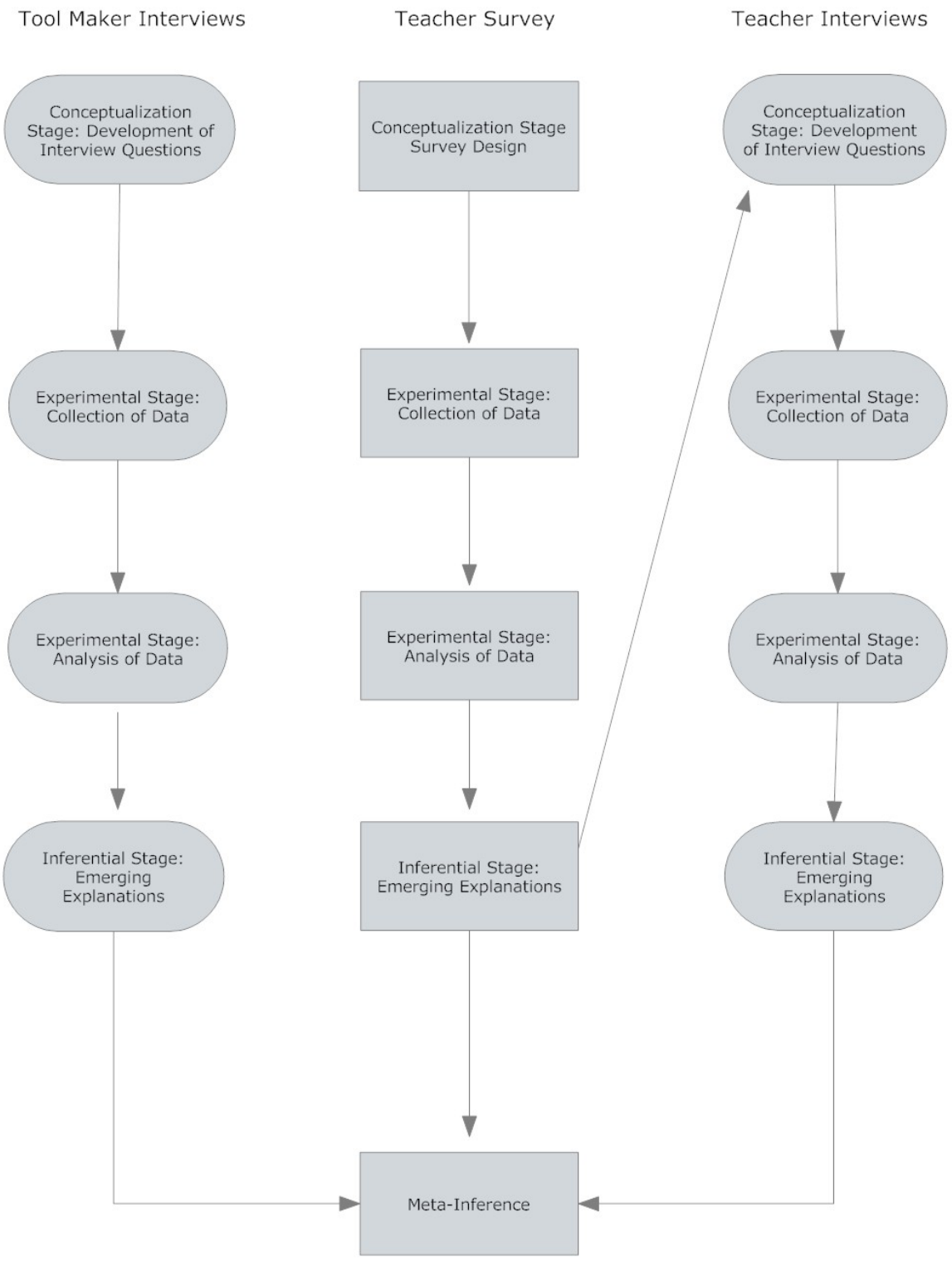

Figure 3: Graphical Illustration of Mixed Research Design (based on the illustrations from Teddlie and Tashakkori) [212]

Following the procedure described by Cohen [46], the study uses multiple sources for data collection gathered in three stages. The first stage surveys teachers who use pedagogical tools in an introductory programming course to identify the tools most commonly used and to explore the characteristics that are common to these tools. Using *progressive focusing*, this study starts by collecting a wide range of data, and then, by sifting, sorting, reviewing and reflecting on them, the salient features of the situation emerge [46, 206]. The data are then used as the agenda for the next stage. Progressive focusing indicates the desire to leave observations and interpretations open [206]. The second stage explores the survey results in more detail through Structured open-ended interviews with teachers that use, have used, or consciously choose not to use tools in their programming courses. This provides more detailed information on the reasons for the use, non-use, and integration of the tools. Finally, to provide an explanatory insight of the tool and the motivation for its creation, semi-structured interviews with several tool developers were conducted. The data analysis shows the commonalities, differences and similarities of the characteristics of pedagogical tools developed for and used in introductory programming classes.

6.3 Stage 1: Survey to Teachers

6.3.1 Introduction

The first of the three stages of this research study employs a Web-based survey to assess the perceptions of teachers who use pedagogical tools in an introductory programming course. The purpose of this inquiry was

- to obtain a detailed list of characteristics common to effective pedagogical tools used in introductory programming courses offered in secondary schools, community colleges, and 4-year colleges and universities
- to make inferences about the characteristics of a useful and/or effective tool in the teaching and learning of programming
- to make inferences about the characteristics of a tool that is considered ineffective or detrimental in the teaching and learning of programming

- to make inferences about the a tool's adoption by introductory programming teachers

The online survey design and implementation followed five methodological components that are critical to successful Web-based surveys [9]:

1. survey design
2. subject privacy and confidentiality
3. sampling and subject selection
4. distribution and response management
5. survey piloting

Each of these components is addressed below with respect to the survey used in this study.

6.3.2 Survey Design and Development

The survey questions were developed using information gathered from the literature review presented in Chapters 4 and 5 of this dissertation. This review presented the goals of tool developers, the results of past research on pedagogical tools, and the results of research focusing on learning styles and teaching pedagogies. Combining empirical results and anecdotal evaluations of previous studies, the goals of tool developers, and this researcher's personal experience, a listing of characteristics that may contribute to a tool's effectiveness (positively or negatively) was developed. From this list, the survey questions were assembled and organized. Table 1 gives an example of a mapping between the goals of Sanders and Dorn [200], the developers of Jeroo, and select survey questions. The complete mapping for the survey questions can be found in Appendix D.

The survey is divided into the ten *pages* briefly described below. The complete survey can be found in Appendix A of this dissertation.

1. General Information: What tool is being used and why was the tool was initially chosen?

Developer's Goals	Survey Questions
easy to use	has an intuitive interface and has a learning curve that is not too steep
engages students immediately	supports students active engagement in learning activities
provides students with a better understanding of programming concepts	supports an understanding of abstract concepts helps understanding of program execution
increases student satisfaction and enthusiasm for programming	Students enjoy using this tool. The initial experience of this tool positively affects subsequent programming courses.

Table 1: Mapping of Developers' Goals with Survey Questions (example)

2. Tool Characteristics: What are the general characteristics believed to contribute to the effectiveness of the tool and which of these characteristics are perceived to be the most significant contributors? The characteristics in this category focus on the user interface of the tool and student interaction.

3. Programming Characteristics: What are the characteristics dealing with programming, testing, debugging, and interaction believed to contribute to the effectiveness of the tool and which of these characteristics are perceived to be the most significant contributors?

4. Learning and Teaching: What are the characteristics relating to student learning and teaching methodology that are perceived to contribute to the effectiveness of the tool and which of these characteristics are believed to be the most significant contributors?

5. Materials and Tool Summary: What are the characteristics specific to auxiliary materials and perception about the use of the tool in the course that contribute to the effectiveness of the tool and which of these characteristics are believed to be the most significant contributors?

6. Negatives: What are the perceived negative aspects (if any) of the tool environment

that hinder either teaching or learning?

7. Training and Experience: What amount of training in the use of the tool was available and/or provided and how much experience does the teacher have using the tool?

8. Your Course: Information was collected about the course in which the tool is being used.

9. About You: Demographic information was collected.

10. More about You: Optional contact information was collected.

In addition to dichotomous questions, the survey includes multiple choice (multiple answers), multiple choice (single answer), and rating (Likert scale) questions. The multiple choice questions allowed for a choice of *other* with an open ended response box. The only question on the survey that *required* an answer was the identification of the tool that was being evaluated.

The survey was physically created using SurveyMonkey [210], a tool that allows for the development and deployment of on-line surveys and the collection and analysis of the results of these surveys.

6.3.3 Subject Privacy and Confidentiality

An Internet survey is naturally suspect of threatening an individual's privacy and the confidentiality of data that are collected through the survey. Privacy refers to the individuals' right to control access to themselves, freedom from unauthorized intrusion [43, 46, 156]. For this study, the initial contact for many of the respondents was a personal email request or an email sent to a listserve whose membership is composed of computing educators. Although the email may have been considered an intrusion, the survey was voluntary and teachers could choose not to participate.

Confidentiality refers to data. It refers to the obligation to inform research subjects how their data will be used. The survey introduction explained the purpose of this research.

Although there were questions that asked for name, affiliation, and email address, these questions were optional and a statement on this page of the survey informed the respondent that this information will be kept private and the respondent will not be identified in any publication or presentation of the study findings.

SurveyMonkey assists with the technical aspects of privacy and confidentiality in the nature of their tool. The following commitment is taken from their product description.

> We employ multiple layers of security to make sure that your account and your data remains private and secure. We employ a third-party firm to conduct daily audits of our security, and your data resides behind the latest in firewall and intrusion prevention technology. For extra sensitive surveys, you can add SSL to your account, so that your data is collected in a totally encrypted environment.
>
> In an age where personal data is traded like a commodity, SurveyMonkey pledges to respect your privacy. Any data that you collect is kept completely and absolutely confidential. We have never been affiliated with any third-parties, and we never accept any advertising [210].

6.3.4 Sampling and Subject Selection

As stated in Chapter 3, this study does not represent a random sampling of teachers. The technique used for sampling was a non-probability method known as *purposive sampling* [46]. This type of sampling selects subjects because of some defined characteristic(s) [173]. In this study, the surveyed teachers satisfied one or more of the following criteria:

- The teacher graded the Advanced Placement Computer Science (AP CS) Examination in June 2008 and expressed a willingness to participate in this study
- The teacher is a member of the AP CS Electronic Discussion Group (AP CS EDG)
- The teacher was an attendee at SIGCSE 2009
- The teacher was a participant in a teacher workshop or summer institute presented by this researcher between 2003 and 2008.

This sample is satisfactory for the needs of this research and has been chosen for a specific purpose of identifying the perceptions these teachers have about pedagogical tools used in introductory programming classes. This sample does not pretend to represent the wider population of all programming instructors or even of all introductory programming instructors. The sample is deliberately selective and biased. Purposive sampling was used in order to access *knowledgeable people*. In this study, the responses of the sample were not taken to generalize to the population but rather to acquire in-depth information from those who are in a position to give it [46]. The teachers surveyed are most likely at a higher level of professional involvement in professional conferences, professional development workshops and activities, and discussion groups focusing on computer science education.

6.3.5 Distribution and Response Management

An Internet survey reduces the time to distribute, gather data, and process the data. The management of the responses was done automatically by SurveyMonkey. This online survey allowed for the respondents to complete it at a time that was convenient to them, minimizing any organizational constraints on the researcher's part.

To simplify the introduction of the survey, and to acquire a response rate, two initial emails were sent. The purpose of the first email was primarily to eliminate from a list of respondents those whose emails bounced. This email informed the recipients that a request to complete a survey would be sent to them soon. This allowed the recipients to respond and request not to be included in the survey. The second email was a detailed explanation of the survey and the survey link.

The individual e-mail solicitations were sent to people whose email addresses were in the researcher's contact list and who satisfied the criteria listed for the survey sample in section 6.3.4. The initial email invitation was sent to 173 computer science educators. The request was for the survey to be completed within four weeks. There were two short reminders sent to those on this individual mailing list that had not responded, one sent

two weeks after the first email and the last sent four days before the requested completion date. Identifying oneself was optional and several teachers completed the survey for more than one tool so the exact response rate from the initial invitation can not be accurately determined. At least 116 of those initially invited (67%) completed the survey between February 3, 2009 and March 1, 2009. There were 166 responses to the initial invitation.

The individual solicitation was followed by solicitation to the Advanced Placement Electronic Discussion Group (mailing list). Approximately 1,950 teachers subscribe to this mailing list. The survey remained open for responses until June 24. Fifty-eight additional surveys were completed. A total of 224 surveys in all were completed from February 2, 2009 to June 24, 2009. Appendix B contains the text of the emails.

6.3.6 Survey Piloting

Following the procedure of Cohen [46], a pre-pilot of the survey was conducted to focus on format and coverage of the survey questions, gaining feedback from a limited number of respondents and experts in the field of computer science education. This pre-pilot study included six programming teachers representing both secondary school teachers and university faculty, some of whom were experienced in question design, survey development, and educational research. The purpose of the pre-pilot was to eliminate ambiguities and difficult wording, to check the clarity of questionnaire items, to gain feedback on the types of questions and their formats, to identify omissions, redundant and irrelevant items, and to check the length of the survey.

Significant revisions were made based on the suggestions of the pre-pilot participants. The survey was then sent to a pilot group of ten additional respondents before the final version of the survey was open. Only minor changes were made as a result of the suggestions from those participating in the pilot study.

6.4 Stage 2: Interviews with Teachers

6.4.1 Introduction

Interviews with teachers were aimed at capturing thoughts and perceptions about the different pedagogical tools used by teachers in introductory programming classes. The interviews are meant to solicit perceptions rather than collect quantitative information. Through interviews, data pertaining to the underlying motivations and implications of using tools were collected.

6.4.2 Interview Design and Development

Each interviewee took part in a one-to-one phone or in-person interview that lasted approximately twenty minutes. The interview was a standardized open-ended interview [46, 212]. In this type of interview, the exact wording and sequence of questions are determined in advance. All respondents are asked the same basic questions in the same order. Questions are worded in a completely open ended format. This interview type was chosen to increase comparability of the responses and to reduce interviewer effects and bias. The interviewer adhered to the following guidelines for conduct of interviews suggested by Cohen [46].

The interviewer followed the guidelines for structured interviews described by Denzin [58]:

- A standardized explanation about the study was given at the beginning of the interview.
- There was no deviation from the interview introduction, sequence of questions, or question wording.
- There were no interruptions during the interview process.
- Answers were never suggested, agreed with, or disagreed with. Personal views of the interviewer were not given.
- Categories were not added and wording of questions was not changed.

The interviewer played a neutral role, not interjecting opinions of the respondent's answers. A balanced rapport was established at the beginning of the interview; the interviewer was friendly and casual but directive and impersonal when asking the interview questions providing a style of interested listening that rewards the respondent's participation but does not evaluate the respondent's answers [58].

The questions for the teacher interviews were based on the responses to the teacher survey. Where the survey asked questions about a particular tool, the interview focused on characteristics that are shared (or not shared) by tools within a category and across different categories. Questions delved into the teacher's motivation for choosing the tool, the motivations for continuing to use the tool (or not), and teacher training in using the tool and integrating the tool into the curriculum. The interviews also queried teachers on the positive and negative aspects of tools they choose to use (or consciously choose not to use) in their introductory programming courses.

6.4.3 Subject Privacy and Confidentiality

The purpose of the interview was explained in an email invitation and again at the beginning of the interview. Since the teacher had volunteered to take part in the interview, the email should not be viewed as an unauthorized intrusion that violates the individual's privacy. All interviews were recorded with the permission of the respondent. This permission was given in an email response confirming the interview and again at the beginning of the interview. The interviewee was encouraged to ask any questions that would clarify any doubts about the interview or the research study. The interviewee was informed that personal names or names of affiliate institutions would not appear in any written report or in any presentation of this study.

6.4.4 Sampling and Subject Selection

There were fifty-one respondents to the teacher survey who indicated using (or choosing not to use) at least five pedagogical tools in their introductory programming classes. The sample for the interviews was purposefully selected from these fifty-one teachers using a

process known as extreme case sampling [48]. Extreme case sampling focuses on cases that are rich in information and do not necessarily display the average teacher's experience with pedagogical tools. This interview sample will highlight information that may be less obvious when looking at teachers having experience with only a small number of tools.

A spreadsheet was created to track tools used by each of the fifty-one individuals. Using this spreadsheet, the data were sorted according to the number of tools used (or consciously chosen not to use). In evaluating the sorted list, interview priority was given to those teachers that ranked high in the sorted list and also ranked high across categories. For example, if a Teacher#1 used six tools but five out of the six tools were IDEs and teacher#2 used six tools where two were microworlds, two were IDEs, one was game making software, and one was a software library, Teacher#2 would be given interview priority over Teacher#1. A list containing the names of eleven teachers that included six high school teachers, four university/college faculty, and one middle school teacher was developed. These eleven teachers were invited, via email invitation, to be interviewed for this study. All eleven accepted.

6.4.5 Interview Process and Response Management

An appointment was made with the interviewees via email. The interview duration varied from 15 to 35 minutes. The interviews were recorded with permission and then transcribed. The interviews were transcribed in a two-step process. The audio tapes were initially transcribed smooth verbatim (crutch words eliminated) by Hi-Tech [97] or Accentance, Inc. [1], both on-line transcription service providers. The transcribed text was then manually reviewed by the researcher against the original audio taped interview for accuracy. The written text was corrected when necessary. When transcribing and analyzing the responses to the survey, the respondents were identified by a unique identification code. To facilitate the coding and analysis, the question data was identified using question number and interviewee identification code. This process allowed for identifying major themes, common issues, and patterns of repeated perspectives both within and across the questions [51, 75, 207]. This study employs a descriptive method of

reporting the interview data.

6.5 Stage 3: Interviews with Tool Developers

6.5.1 Introduction

Interviews with selected tool developers or members of the tool development team were aimed at capturing the developer's pedagogical goals in the tool's design and development. The interviews are meant to solicit information about the pedagogical tool, offering a different perspective on the tool use and the tool adoption in the introductory programming course. Does the tool developer have the same goals in designing the tool as the teachers have when adopting the tool? Through interviews with tool developers, data pertaining to the underlying motivations and implications of the tool use were collected.

6.5.2 Interview Design and Development

Each interviewee took part in a one-to-one phone or in-person interview that lasted approximately twenty minutes. The interviews with the tool developers were semi-structured interviews [70]. In this type of interview, the overall structure was planned by the researcher in advance, with a script of main questions. The order of the questions was altered to adapt to the respondent's answers. The respondent was given considerable freedom of expression but the interviewer controlled the interview to ensure coverage. The interviewer played a neutral role. This type of interview allowed the researcher to obtain specific qualitative information from the respondent, to obtain general information relevant to specific issues dealing with pedagogical tools in introductory programming courses, and to gain a range of insights dealing with the design, development, and use of the interviewee's specific pedagogical tool.

The questions for the toolmaker interviews were designed to investigate the motivation behind the creation of the tool, the tool developers' perceived value of the tool, and the toolmakers intended use for the tool. There is a wide-range of pedagogical tool use in introductory programming courses. The researcher is seeking information from the toolmakers that will add insight about the characteristics of successful tools and about the

successful methods of tool dissemination.

6.5.3 Subject Privacy and Confidentiality

The purpose of the interview was explained at the beginning of the interview. All interviews were recorded with the permission of the respondent. The interviewee was encouraged to ask any questions that would clarify any doubts about the interview or the research study. The interviewee was informed that personal names would not appear in any written report or in any presentation of this study. All interviewees granted permission for the tool name to be used in the discussion of the interviews and analysis of the results.

6.5.4 Sampling and Subject Selection

A purposive convenience sampling was used for the tool developers interviews. Only tool developers of pedagogical tools used in introductory programming classes and tools that were listed on the teacher survey were considered. From the list of 29 tools on the survey, six tool developers were invited to be interviewed. All accepted. The interviewees were primary developers or members of the development teams of the following tools: Alice, BlueJ/Greenfoot, Java Task Force Library, Jeroo, Karel J. Robot, and ObjectDraw. Since the researcher and many of the tool developers were in attendance at the Special Interest Group in Computer Science Education (SIGCSE) 2009 Symposium in March 2009, most of the interviews with the developers were conducted at the symposium.

6.5.5 Interview Process and Response Management

The interview duration varied from 10 to 25 minutes. As stated above, most of the interviews were conducted at the Special Interest Group in Computer Science Education (SIGCSE) 2009 Symposium in March 2009. Two of the interviews were conducted via telephone and one interview was conducted in-person at a pre-arranged interview appointment. The interviews were recorded with permission and then transcribed. As with the teacher interviews, the recorded interviews with the tool developers were transcribed in a two-step process. The audio tapes were initially transcribed smooth verbatim (crutch words eliminated) by Accentance, Inc. [1], an on-line transcription service provider. The

transcribed text was then manually reviewed by the researcher against the original audio taped interview for accuracy. The written text was corrected when necessary. When transcribing and analyzing the responses to the survey, the respondents were identified by a unique identification code. To facilitate the coding and analysis, the question data was identified using question number and interviewee identification code. This process allowed for identifying major themes and sub themes both within and across the questions [51, 207]. This study employs a descriptive method of reporting the interview data.

6.6 Summary

By employing a mixed methods approach that implements both a sequential method design (survey followed by teacher interviews) and a parallel method design (tool developer interviews), information about tool characteristics, tool adoption, and information dissemination is collected. This research design yields results that provide information to tool developers when developing or modifying tools and provides a reference to teachers choosing pedagogical tools to integrate into their introductory programming courses.

Chapter 7

Results

7.1 Introduction

The primary goal of this study is to discover characteristics that teachers perceive contribute to the effectiveness of a pedagogical tool used in introductory programming courses. The purpose of this study is to provide guidance to teachers when evaluating tools for introductory programming classes and to provide tool developers with feedback from the teachers who use these tools with hopes that future tools will be designed and developed to meet teacher and student needs in the introductory programming classes.

By analyzing the responses to the survey and interviews, the hypotheses that were presented in section 3.5 will be addressed. The results of this study are based on the perceptions of computer science educators:

1. Using pedagogical tools in introductory programming classes eases and promotes the learning of programming.

2. Using pedagogical tools in introductory programming classes eases and promotes the teaching of programming.

3. Effective pedagogical tools used in introductory programming classes have common characteristics.

4. Tools that teachers consciously choose NOT to use in introductory programming classes have common characteristics.

5. Teachers initially choose to use a pedagogical tool because of a perceived task-technology "fit."

This chapter reports the results of the survey and interviews involved in this study. The results of this study provide perceptions of a limited population of introductory

programming teachers. The teachers surveyed are most likely at a higher level of professional involvement in professional conferences, professional development workshops and activities, and discussion groups focusing on computer science education than the general population of introductory programming teachers. Chapter 8 discusses the survey results together with the interview data and Chapter 9 summarizes the results in addressing each hypothesis.

7.2 Teacher Survey Results

The purpose of the teacher survey was

- to obtain a detailed list of characteristics common to effective pedagogical tools used in introductory programming courses offered in secondary schools, community colleges, and 4-year colleges and universities
- to make inferences about the characteristics of a useful and/or effective tool in the teaching and learning of programming
- to make inferences about the characteristics of a tool that is considered ineffective or detrimental in the teaching and learning of programming
- to make inferences about the a tool's adoption by introductory programming teachers

The complete survey can be found in Appendix A. Section 6.3 describes the procedure used to formulate the survey questions.

A completed survey depicts the teacher's perceptions about one specific tool. If the teacher has experience with more than one tool, the completion of multiple surveys was requested, one survey for each tool used. The question format for identifying the tool being evaluated provided a drop-down menu with a choice of twenty-nine tools plus a choice of *other* where the name of the tool was supplied by the respondent. The tools evaluated by teachers completing this survey appear in Table 2. There were 19 tools included in the pull-down menu that were not evaluated by any respondent. These tools

are: Buggles and Bagels, CodeWiz, Greeps, JamTester, ACM Java Task Force Graphics, Jeliot 3, JHAVE, JPie, Java Power Tools, PigWorld, Robotran, XTango, and Zeus. A total of 224 surveys were collected from at least 121 different individuals. Since providing a name was optional, the exact number of distinct individuals can not be determined.

Tool Being Evaluated	Percent Responding	Number of Responses
Alice	21.4%	48
BlueJ	21.0%	47
Eclipse	8.0%	18
JCreator	7.6%	17
GridWorld	4.9%	11
jGRASP	5.4%	12
Karel J. Robot	4.0%	11
DrScheme	3.6%	8
NetBeans	3.1%	7
DrJava	2.7%	6
JUnit	2.2%	5
ObjectDraw	2.2%	5
Jeroo	1.8%	4
Greenfoot	1.8%	4
GameMaker	0.9%	2
Raptor	0.9%	2
Scratch	0.9%	2
MediaComp	0.4%	1
Lego Mindstorms	0.4%	1
other	5.8%	13
answered question		**224**
skipped question		**0**

Table 2: Tools evaluated through this survey by teachers of introductory programming classes

The choice of 'other' allowed for a write-in response. These responses included: Stagecast Creator, Javabat, Visual Basic, Visual C++, IDLE, Textpad, pcGrasp, Visual Studio, Visual C#, GIMP, Professor J, and Intellibrain with only Visual Basic being listed multiple times (3). The results presented in Table 2 noticeably identify two tools, Alice and BlueJ, with much higher response rates than the others. This does not translate into these two tools being used more than others. Table 2 simply reports that more teachers chose to *evaluate* Alice and BlueJ than evaluating the other tools they may or may not be using in their classes. To obtain a better picture of what tools are being used or chosen not to be used in introductory programming courses, the responses of the 121 teachers who chose to identify themselves are summarized in Table 3. This summary is the result of creating a spreadsheet of tool use where each teacher was listed along with the tools used or chosen not to be used by that teacher. By considering only the 121 identified survey responses, there is no duplication of data and an accurate account of tool *use* for those 121 teachers is obtained.

The results of the survey will be reported first by looking at the cross category summary of the survey responses and then by looking at the aggregation of the results of the individual tool categories described in Section 3.1. The presentation of the results is divided into several subsections, each referencing a different page (focus) of the survey.

1. General information about choosing to use the tool
2. Overall tool characteristics
3. Tool characteristics focusing on programming environment, testing, and interaction
4. Tool characteristics contributing to the learning to and the teaching of programming
5. Materials, support, and teacher perception of tool use
6. Perceived negative characteristics of the tool
7. Training and Experience of the respondent
8. Demographics of respondent

Primary Tool Category	Tool	Number choosing to use tool	Number choosing not to use tool	Total
Microworlds	GridWorld	51	2	53
	Alice	41	8	49
	Karel J. Robot	25	11	36
	GreenFoot	9	4	13
	Jeroo	7	2	9
IDEs	BlueJ	48	14	62
	Eclipse	34	13	47
	JCreator	26	10	36
	DrJava	12	5	17
	DrScheme	7	10	17
	jGRASP	10	5	15
	NetBeans	10	4	14
	JBuilder	0	2	2
Libraries	Object Draw	13	2	15
	MediaComp	5	1	6
	JTF	1	0	1
Visualization Tools	Jeliot 3	2	1	3
	Raptor	2	0	2
Robots	Lego Mindstorms	13	4	17
Game Making	Game Maker	3	1	4
Other	JUnit	12	0	12
	Scratch	7	1	8
	JavaBat	2	0	2
	JamTester	4	0	4

Table 3: Tools chosen to be used or chosen not to be used by teachers

Of the 224 completed surveys, 83% evaluated a tool presently being used in the class (users) while the remaining 17% evaluated tools that are not being used (non-users). The non-users have consciously chosen not to use the tool and may or may not have previously used the tool. The cross category summary will report the results of three groups, the total, the users, and the non-users.

Providing demographic information was optional. Of the 224 completed surveys, 100 were identified as being completed by males and 81 were identified as being completed by females. The cross category summary will report the results of both genders when they differ from the total results.

Introductory programming courses are taught in middle schools, high schools, and in colleges. Of the 224 completed surveys, 129 were completed by those teaching in secondary schools (grades 7 - 12) and 59 were completed by those teaching in a two- or four- year college. The cross category summary will report the results of the two groups, secondary school teachers and those teaching in colleges or universities, when they differ from the total results.

The aggregation summary will report the results for each of the questions by tool category: Microworlds, Libraries, IDEs, and Visualization Tools.

7.2.1 Survey Question Types

The survey contained the following question types:

- Choice Questions: Multiple Answers

  Many of the survey questions direct the respondents to check items that satisfied certain criteria chosen from a list of given items. The respondent could choose to check zero or more items from this list. The respondent could select a choice of *other* to write-in a response that was not included in the given list. Since the purpose of this study is to determine the characteristics that teachers perceive to contribute to the effectiveness of a tool, the results summary identifies the responses

that are chosen in a majority of the completed surveys. A majority is a subset that is more than 50% of the group responding. If no item was chosen by a majority, the top three items are listed.

- Choice Questions: Single Answer

  Many of the survey questions direct the respondents to check only one item from a given list of items or from the list of items predetermined from the respondent's answers to a multiple answers question. The results list those items that were chosen by more than 50% of the total surveys completed. If no item was chosen by a majority, the top three items are listed.

- Ranking Questions

  Many of the survey questions ask respondents to rank the top three items, in order of importance, chosen from a list of items. This study uses the Borda method [228] to determine the top three items ranked by the survey responses. This method of partial ranking is explained in Section 7.2.2. The results will list the top three ranked characteristics as determined by the group being surveyed.

- Dichotomous Questions

  This is a question in which the respondent chooses one of two possible responses. The results will be presented as the percentage of completed surveys choosing each answer.

- Likert Scale Questions

  This survey included several 5-point Likert scale questions with answer choices ranging from 'strongly agree' to 'strongly disagree'. The respondents are asked to specify their level of agreement to a statement. The data are summarized by presenting the descriptive statistics that includes the frequency count for each choice, the median, the mode, the mean, and standard deviation.

- Open Ended

  The survey included a few open-ended questions asking the respondents for

additional information. The results are presented in list format.

7.2.2 Analysis of Rank Data

Many of the survey questions asked teachers to rank the top three characteristics in order of importance. This study uses the Borda method to determine the top three characteristics ranked by the survey responses. The Borda method has several advantages including simplicity, computational efficiency, and a minimum of voting paradoxes, but it also has some inherent limitations: (1) It is defined for equal attribute weight only; and (2) it does not handle inexact, uncertain and fuzzy data [134]. The data collected by this survey tool is ordinal in nature and for the purpose if this study, the ranking of the ordinal values is sufficient thus minimizing the limitation of (1). The choices are discrete choices so (2) does not negatively affect the results when this method is applied.

The Borda method is based on an algorithm for priority ordering and is used in this study as a ranking method in which a list of characteristics is ranked based on information gathered from the survey responses. The method used is based on algorithms explained and used in various studies involving ranked and partially ranked data [63, 113, 134, 164]. The specific approach and algorithm used is explained below.

Suppose there are N survey responses and K characteristics. A preference order of characteristics is supplied by each survey response ranking the top three characteristics as most important, second most important, and third most important. The rankings *4-th* to *k-th* are considered to be ranked equally (non-rankings) in this study. For each survey response, points K - 1, K - 2, and K - 3 are assigned to the first-ranked, second-ranked, third-ranked characteristics respectively. For example, if there is a total of 14 characteristics, 13 points are assigned to the first-ranked, 12 points are assigned to the second-ranked, and 11 points are assigned to the third-ranked characteristic. The number of points assigned to a particular characteristic is the number of characteristics ranking below it. The final preference ordering of the top three characteristics is determined to be the three characteristics with the highest total points from all survey responses. In other

words, if r_{ij} is the rank of the *i-th* characteristic of the *j-th* response, the Borda count, b_i, for this *i-th* characteristic is

$$b_i = \sum_{j=1}^{N} (K - r_{ij}).$$

The characteristics are then listed in decreasing order of the Borda counts. Ties are handled by evaluating the rank for a tied alternative as the average of the associated rankings [134]. Since the non-ranked characteristics are considered equal, the rank associated with each is

$$r_i = \frac{1}{K-3} \sum_{j=0}^{K-4} j = \frac{K-4}{2},$$

the average of the associated rankings. Ranks were assigned to each characteristic in each survey response. This is essential to avoid any plurality pitfalls [164].

7.2.3 Cross Categories Summary

The cross categories summary is reported with no consideration given to the individual tool or specific category of the tool being evaluated. All responses to each question are included in this summary of the results.

The purpose of this study is to determine the common characteristics perceived by teachers to contribute to the effectiveness of pedagogical tools used in introductory programming classes. All tools do not necessarily share all characteristics. A particular characteristic may simply not be applicable to the tool and its primary purpose. As examples, the IDE, Eclipse, does not involve storytelling; Jeroo, by design, does not does not provide for support of data types, variables, or user-provided input [200]. Tool developers may have consciously chosen not to incorporate certain aspects into their tools. However, certain characteristics may be perceived as important enough to teachers that, if available in a tool, would increase the effectiveness of the tool in an introductory programming class. Several questions in the survey asked teachers to list, in priority order, those characteristics (if any) that the tool *should* support thus resulting in improving the effectiveness of the tool. The parenthesized "if any" may have led to lower

response rates for this question. In the case where the tool was considered to include all necessary characteristics, this question would not have been answered.

7.2.3.1 General Information About Choosing to Use the Tool

The first section of the survey identified the tool being evaluated and the reasons for the teacher's initial decision to use this tool in an introductory programming course.

Reasons for initially choosing to use this tool in an introductory programming course

This question was answered by 99% of those being surveyed. The question allowed for multiple answers. Of the eight given reasons for initially choosing to use this tool in an introductory programming course, the ones chosen in more than 50% of the completed surveys were:

- to introduce programming in a more enjoyable way (65.8%)
- recommended to me by other computer science educators (60.8%)

The responses of the users and the non-users agreed with the total responses in identifying these two reasons with a majority of the non-users including a third reason,

- to attract a diverse group of students (56.8%)

Males and females responded to this question with the same choices as the total, as did the secondary school teachers. Those teaching in colleges and universities differed in the response to this question. The reasons for initially choosing this tool identified by more than 50% of the college surveys completed were:

- to introduce programming in a more enjoyable way (66.7%)
- task-technology fit (54.4%)
- tool complements my learning style (52.6%)

When asked to identify the most influential reason for choosing to use this tool in an introductory programming course,

- to introduce programming in a more enjoyable way

was selected most often by all groups: total survey responses, users, non-users, males, females, secondary school teachers, and college teachers.

7.2.3.2 Overall Tool Characteristics

This section queried teachers about the general tool characteristics that are perceived to be significant contributors to the effectiveness of the tool.

Characteristics of the tool that are perceived to contribute to the tool's effectiveness

The characteristics in this category focus on the user interface of the tool and student interaction. Multiple answers were permitted. The question was answered in 91% of the total surveys completed. The characteristics identified to contribute to the effectiveness of the tool in more than 50% of those surveys were:

- improves first-time programmer's experience (75.4%)
- has a learning curve that is not steep (74.4%)
- supports an interactive environment (71.9%)
- has an intuitive interface (62.6%)
- is flexible (allows use at many different levels of learning) (59.1%)
- embodies ideas that support core programming concepts (57.1%)
- includes graphical components (54.2%)

The majority choices of the users were the same as the majority choices of the total survey responses. The non-users list of majority choices did not include:

- is flexible (allows use at many different levels of learning) (42.4%)
- embodies ideas that support core programming concepts (45.5%)

The majority choices of the males and the majority choices of the females agreed with the majority choices of the total survey responses as did the majority choices of the the secondary school teachers. While the responses of the college teachers agreed with most of the choices of the total survey responses,

- includes graphical components (43.9%)

was not a majority choice of the college responses.

There was an 88% total response rate when asked to rank, in order of priority, the top three of the characteristics contributing to the effectiveness of the tool. The responses of the users (92% response rate) agreed with the total survey responses. The responses of the non-users (76% response rate) differed. Table 4 lists, in order of priority, the rankings of the top three characteristics perceived to contribute to the effectiveness of the tool in an introductory programming class as determined by the total survey responses, the responses of the users, and the responses of the non-users.

Rank	Total	Users	Non-Users
1	improves first-time programmer's experience	improves first-time programmer's experience	supports an interactive environment
2	has a learning curve that is not steep	has a learning curve that is not steep	improves first-time programmer's experience
3	has an intuitive interface	has an intuitive interface	includes graphical components

Table 4: Top three characteristics, ranked in priority order, that are perceived to contribute to the tool's effectiveness as determined by the total survey responses, the responses of the users, and the responses of the non-users

When comparing the results by gender, the females responses (92.5% response rate) agreed with the top three rankings of the total. The male responses (99% response rate) ranked the characteristics contributing to the effectiveness of the tool slightly different from the rankings of the total. In order of priority, the male and female rankings of the

top three characteristics perceived to contribute to the effectiveness of the tool in an introductory programming class are listed in Table 5.

Rank	Total	Males	Females
1	improves first-time programmer's experience	improves first-time programmer's experience	improves first-time programmer's experience
2	has a learning curve that is not steep	has a learning curve that is not steep	has a learning curve that is not steep
3	has an intuitive interface	supports an interactive environment	has an intuitive interface

Table 5: Top three characteristics, ranked in priority order, that are perceived to contribute to the tool's effectiveness as determined by the total survey responses, the male responses, and the female responses

The responses of the college teachers (100% response rate) and the responses of the secondary school teachers (94.5% response rate) agreed the total survey responses in the ranking of the top three characteristics perceived to contribute to the effectiveness of the tool.

Characteristics not adequately supported by the tool (if any such characteristics are perceived to exist)

This study is concerned with the characteristics that teachers perceive to contribute to the effectiveness of a tool. It is also concerned with the characteristics that teachers perceive to be inadequately supported by the tool. When asked to rank, in priority order, the characteristics that this tool should more adequately support (if there are any such characteristics), there was a 36% total response rate, a 35% users response rate, and a 43% non-users response rate. Table 6 lists the characteristics that were identified as not being adequately supported by the tool. The parenthesized "if any" may have led to lower response rates for this question. In the case where the tool was considered to include all necessary characteristics, this question would not have been answered. The two characteristics ranked third in Table 6 for the total respondents had equal Borda numbers.

Rank	Total	Users	Non-Users
1	allows for easy transition to a more robust programming environment	allows for easy transition to a more robust programming environment	allows for easy transition to a more robust programming environment
2	is flexible(allows use at many levels of learning	is flexible(allows use at many levels of learning	improves a first-time programmer's experience
3	supports abstraction improves a first-time programmer's experience	supports abstraction	supports a concepts-first approach

Table 6: Characteristics, in priority order, that are perceived to be inadequately supported by the tool as determined by the total survey responses, the responses of the users, and the responses of the non-users

Table 7 compares male (48% response rate) and female (28.4% response rate) rankings of the characteristics that are perceived to be inadequately supported by the tool (if there are any such characteristics). The two characteristics ranked third for the total had equal Borda numbers as did the two characteristics ranked third for the female responses

Rank	Total	Males	Females
1	allows for easy transition to a more robust programming environment	allows for easy transition to a more robust programming environment	allows for easy transition to a more robust programming environment
2	is flexible(allows use at many levels of learning)	improves first-time programmer's experience	incorporates storytelling
3	supports abstraction improves a first-time programmer's experience	has an intuitive interface	is flexible (allows use at many levels of learning) supports abstraction

Table 7: Characteristics, in priority order, perceived to be inadequately supported by the tool as determined by the total survey responses, the responses of the males, and the responses of the females

The characteristics perceived to be inadequately supported by the tool (if there are any such characteristics) by those teaching in college (33% response rate) and those teaching in secondary schools (41% response rate) are listed, in priority order, in Table 8. The two

characteristics ranked third for the total respondents had equal Borda numbers.

Rank	Total	College Teachers	Secondary School Teachers
1	allows for easy transition to a more robust programming environment	allows for easy transition to a more robust programming environment	allows for easy transition to a more robust programming environment
2	is flexible(allows use at many levels of learning)	has a learning curve that is not steep	supports abstraction
3	supports abstraction improves a first-time programmer's experience	improves a first-time programmer's experience	includes graphical components

Table 8: Characteristics, in priority order, perceived to be inadequately supported by the tool as determined by the total survey responses, the responses of the college teachers, and the responses of the secondary school teachers

7.2.3.3 Tool Characteristics Focusing on Programming Environment, Testing, Debugging, and interaction

This section queried teachers about characteristics pertaining to the programming environment, debugging, testing, and interaction.

Characteristics related to the programming environment that contribute to the effectiveness of this tool

This question focusing on programming environment characteristics was answered by 85% of the total surveys completed, 86% of the users, 81% of the non-users, 100% of the male responses, 93.8% of the female responses, 100% of the responses of college teachers, and 96% of the responses of secondary school teachers. The only two characteristics dealing with programming environment that were identified to contribute to the effectiveness of the tool by more than 50% of the total survey responses are:

- simplifies the mechanics of programming (53.5%)
- supports visualization of OO concepts (50.1%)

The users majority choices agree with the total choices. The non-users, male responses, and college teachers' responses identified only one majority choice:

- simplifies the mechanics of programming (50%, 59%, 63.2%, respectively)

The only majority choice of the female responses was:

- supports visualization of OO concepts (53.9%)

The programming environment characteristics identified to contribute to the effectiveness of the tool by more than 50% of the responses of the secondary school teachers were:

- supports visualization of OO concepts (53.3%)
- provides multiple views at once (e.g. code window, animation, state) (51.6%)

Characteristics that relate to testing, debugging, and interaction that contribute to the effectiveness of this tool

This question focusing on debugging, testing, and interaction was answered on 82% of the total survey responses, 85% of the users' responses, and 68% of the non-users' responses. The characteristics that relate to testing, debugging, and interaction identified to contribute to the effectiveness of this tool by more than 50% of the total survey responses are:

- supports comments and documentation (54.1%)
- supports debugging (53.6%)
- supports developing and testing of individual components (51.9%)
- gives immediate feedback about errors (50.3%)
- supports step-by-step evaluation of single programming statements (50.3%)

The majority results for the users and the male responses (95% response rate), were the same as for the total. The was no characteristic relating to testing, debugging, and interaction identified to contribute to the effectiveness of this tool by more than 50% of the non-users.

The only characteristic relating to testing, debugging, and interaction identified to contribute to the effectiveness of this tool by more than 50% of the female responses (91% response rate) was:

- supports comments and documentation (51.4%)

More than 50% of the college teachers' responses (95% response rate) identified

- supports developing and testing of individual components (68.5%)
- supports debugging (61.1%)
- supports comments and documentation (57.4%)
- supports step-by-step evaluation of single programming statements (53.7%)

as contributing to the effectiveness of the tool while the majority of the responses of the secondary school teachers (92% response rate) identified

- supports comments and documentation (53.8%)
- provides meaningful error messages (53%)
- supports debugging (50.4%)
- gives immediate feedback about errors (50.4%)

as characteristics relating to testing, debugging, and interaction that contribute to the effectiveness of the tool.

A 77% total response rate identified the ranking of the top three characteristics, in order of priority, pertaining to the programming environment, debugging, testing, and interaction that contribute to the effectiveness of the tool. The rankings of the users (81% response rate) agreed with that of the total survey responses. The non-users (62% response rate) had a slightly different ranking. Table 9 compares the ranking results of the total, the users, and the non-users. The Borda numbers were equal for the three characteristics ranked third by the non-users.

Rank	Total	Users	Non-Users
1	supports visualization of OO concepts	supports visualization of OO concepts	simplifies the mechanics of programming
2	simplifies the mechanics of programming	simplifies the mechanics of programming	prevents syntax errors
3	supports developing and testing of individual components	supports developing and testing of individual components	supports the visualization of state changes supports visual representation of general programming concepts supports the visualization of program state

Table 9: Top three ranked characteristics, in priority order, that relate to testing, debugging, and interaction that contribute to the effectiveness of this tool as ranked by the total survey responses, the responses of the users, and the responses of the non-users

The top three characteristics pertaining to the programming environment, debugging, testing, and interaction that are perceived to contribute to the effectiveness of the tool as ranked by male responses (96% response rate) and the female responses (79% response rate) are compared to the rankings of the total survey responses in Table 10.

Rank	Total	Males	Females
1	supports visualization of OO concepts	simplifies the mechanics of programming	supports visualization of OO programming
2	simplifies the mechanics of programming	supports visualization of OO concepts	supports developing and testing of individual components
3	supports developing and testing of individual components	supports developing and testing of individual components	prevents syntax errors

Table 10: Top three ranked characteristics, in priority order, that relate to testing, debugging, and interaction that are perceived to contribute to the effectiveness of this tool as ranked by the total survey responses, male responses, and female responses

The top three characteristics that relate to testing, debugging, and interaction that are perceived to contribute to the effectiveness of this tool as ranked by the responses of the

college teachers (96.5 response rate) and the responses of the secondary school teachers (84.2% response rate) are compared to the rankings of the total survey responses in Table 11.

Rank	Total	College Teachers	Secondary School Teachers
1	supports visualization of OO concepts	supports developing and testing of individual components	supports visualization of OO programming
2	simplifies the mechanics of programming	simplifies the mechanics of programming	simplifies the mechanic of programming
3	supports developing and testing of individual components	supports incremental development	provides multiple views at once (e.g. code window, animation state)

Table 11: Top three characteristics that relate to testing, debugging, and interaction that are perceived to contribute to the effectiveness of this tool as ranked by total survey responses, responses of college teachers, and responses of secondary school teachers

Characteristics dealing with programming environment, debugging, testing, and interaction that are not adequately supported by the tool (if any such characteristics exist)

44% of the total survey responses identified characteristics that are perceived to be inadequately supported by the tool. Table 12 lists the top three characteristics (in priority order) ranked by the total survey responses, the responses of the users, and the responses of the non-users that are inadequately supported by the tool. The users' responses had a 55% response rate and the non-users' responses had a 41% response rate to this question. The Borda numbers were equal for the two characteristics ranked third by the non-users.

Table 13 lists the characteristics, in priority order, that are perceived to be inadequately supported by the tool as identified by the male responses (61% response rate) and female responses(38% response rate) compared to the rankings of the total survey responses.

Table 14 compares the rankings identified by the total survey responses to the responses of the college teachers (59.6% response rate) and the responses of the secondary school teachers (46% response rate).

Rank	Total	Users	Non-Users
1	provides meaningful error messages	provides meaningful error messages	supports visualization of OO concepts
2	supports visualization of OO concepts	supports step-by-step evaluation of single programming statements	supports visualization of program code
3	supports step-by-step evaluation of single programming statements	supports an easy way of including event-driven programming	supports visual representation of general programming concepts supports debugging

Table 12: Characteristics, in priority order, that are perceived to be inadequately supported by the tool as determined by the total survey responses, the responses of the users, and the responses of the non-users

Rank	Total	Males	Females
1	provides meaningful error messages	prevents syntax errors	provides meaningful error messages
2	supports visualization of OO concepts	provides meaningful error messages	supports an easy way of including event-driven programming
3	supports step-by-step evaluation of single programming statements	supports visualization of program code	supports visualization of OO concepts

Table 13: Characteristics, in priority order, that are perceived to be inadequately supported by the tool as determined by the total survey responses, the male responses, and the female responses

7.2.3.4 Tool Characteristics Contributing to the Learning of and the Teaching of Programming

This section queried teachers about characteristics pertaining to the learning of and teaching of programming. Approximately 80% of the total survey responses identified characteristics that contribute to the tool's effectiveness as they relate to the students' learning to program.

Characteristics identified to contribute to the tool effectiveness as they relate to the student's learning to program.

The characteristics that are perceived to contribute to student learning identified by more

Rank	Total	College Teachers	Secondary School Teachers
1	provides meaningful error messages	supports an easy way of including event-driven programming	provides meaningful error messages
2	supports visualization of OO concepts	provides meaningful error messages	prevents syntax errors
3	supports step-by-step evaluation of single programming statements	supports visualization of OO concepts	supports step-by-step evaluation of single programming statements

Table 14: Characteristics relating to programming environment, debugging, testing, and interaction, in priority order, that are perceived to be inadequately supported by the tool as ranked by total survey responses, the responses of the college teachers, and the responses of the secondary school teachers

than 50% of the surveys that had answers to this question include:

- supports students' active engagement in learning activities (72.8%)
- engages students of different ability levels (64.4%)
- helps understanding of programming execution (60.6%)
- supports the understanding of abstract and complex concepts (54.4%)
- encourages learning through discovery (53.3%)

The users' (83% response rate) majority choices and the college teachers' (95% response rate) majority choices agreed with the characteristics identified by the majority of the total responses. The male (96% response rate) majority choices agreed with the characteristics chosen by the majority of the total survey responses except did not include:

- encourages learning through discovery (47.9%)

The female (93% response rate) majority choices and the secondary school teachers (94% response rate) majority choices agreed with the characteristics identified by the majority of the total survey responses and also included:

- allows students to work at levels of their choice (57%)

The non-users' majority choices (73% response rate) identified only two characteistics contributing to student learning:

- supports students' active engagement in learning activities (70.4%)
- encourages learning through discovery (63%)

Characteristics identified to contribute to the tool's effectiveness as it relates to the teaching of programming

Characteristics perceived to contribute to the tool's effectiveness as they relate to the teaching of programming were identified by 79% of the total survey responses. Characteristics that were identified by more than 50% of those responding to this question include:

- is a good tool for classroom demonstrations (80.8%)
- can be used for the duration of the course (68.4%)

The users (82% response rate), the females (89% response rate), the males (96% response rate), the college teachers (96% response rate), and secondary school teachers (91% response rate) answered this question with the same majority results as the total.

The non-users' (73% response rate) majority chose only:

- is a good tool for classroom demonstration (73.1%)

as contributing tot he effectiveness of the tool as it relates to teaching.

A 75% total response rate ranked the top three characteristics related to the learning and the teaching of programming. These results are compared to the rankings of the users (79% response rate) and the non-users (59% response rate) in Table 15. The Borda numbers were equal for the two characteristics ranked first by the non-users.
Table 16 identifies the top three characteristics that are perceived to contribute the most to the learning of and teaching of programming. The responses of the males (98% response rate) and the responses of the females (78% response rate) are compared to the total. The characteristics identified in Table 16 are listed in priority order.

Rank	Total	Users	Non-Users
1	supports students' active engagement in learning activities	supports students' active engagement in learning activities	supports students' active engagement in learning activities engages students of different ability levels
2	can be used for the duration of the course	can be used for the duration of the course	encourages learning through discovery
3	engages students of different ability levels	supports the understanding of abstract and complex concepts	

Table 15: Characteristics that are perceived to support the learning and teaching of programming as ranked, in priority order, by the total survey responses, the responses of the users, and the responses of the non-users

Table 17 lists the rankings of the characteristics that are perceived to contribute the most to the learning of and teaching of programming as identified by the responses of the college teachers (98% response rate) and the responses of the secondary school teachers (84% response rate). These rankings are compared to the rankings of the total survey responses. The characteristics identified in Table 17 are listed in priority order.

7.2.3.5 Materials, Support, and Teacher Perception of Tool Use

Teachers choose to use a tool for a variety of reasons. One of the reasons may be related to the auxiliary materials, teaching support, and technical support that is available.

Auxiliary materials and support

81% of the total survey responses identifdied characteristics specific to auxiliary materials and support that are perceived to contribute to the effectiveness of the tool. The following lists the characteristics chosen by the majority of the total respondents.

- This tool can be used independent of the textbook chosen. (84.5%)
- Materials about this tool can be found through on-line searches. (70.2%)

Rank	Total	Males	Females
1	supports students' active engagement in learning activities	supports students' active engagement in learning activities	supports students' active engagement in learning activities
2	can be used for the duration of the course	supports the understanding of abstract and complex concepts	can be used for the duration of the course
3	engages students of different ability levels	can be used for the duration of the course	encourages learning through discovery

Table 16: Characteristics that are perceived to support the learning and teaching of programming as ranked, in priority order, by the total survey responses, the male responses, and the female respondents

Rank	Total Respondents	College Teachers	Secondary School Teachers
1	supports students' active engagement in learning activities	supports students' active engagement in learning activities	supports students' active engagement in learning activities
2	can be used for the duration of the course	can be used for the duration of the course	can be used for the duration of the course
3	engages students of different ability levels	is a good tool for classroom demonstrations	engages students of different ability levels

Table 17: Characteristics that are perceived to support the learning to and teaching of programming as ranked, in priority order, by the total responses, the responses of the college teachers, and the responses of the secondary school teachers

- Tutorials on using the tool are provided by authors or community. (63.5%)
- Technical support is provided by authors or community. (63.0%)
- Textbooks that incorporate this tool are available. (54.7%)

The responses of the non-users (81% response rate), the males (94% response rate), the female (99% response rate), and the college teachers (100% respnse rate), identified the same as characteristics relating to auxiliary materials and support as the total survey responses. The responses of the users (82% response rate) and the secondary school teachers (91% response rate) agreed with the total survey responses and also included the

additional characteristic

- Instructor resources (PP presentations, sample problems and labs, syllabus) are provided by authors or community. (51.3% (users), 52.8% (secondary school teachers))

Overall perception of the tool

Teachers were asked for overall perceptions of tool use in an introductory programming course. This question was answered in 75% of the total survey responses.

The characteristics describing the teacher's overall perceptions of the tool chosen by more than 50% of those responding included:

- Students enjoy using this tool. (77.2%)
- Using this tool eases and promotes the learning of programming. (76.6%)
- Using this tool eases and promotes the teaching of programming. (73.1%)
- After using this tool, your students are ready to extend their knowledge by continuing on in a computer science curriculum. (72.5%)
- The initial experience with this tool positively affects subsequent programming experiences. (65.3%)

The majority choices of the users (78% response rate), the males (94% response rate), the females (88% response rate), the college teachers (93% response rate), and the secondary school teachers (91% response rate) agreed with the majority choices of the total survey responses.

The non-user (62% response rate) majority identified three characteristics as contributing to the effectiveness of the tool:

- Students enjoy using this tool. (78.3%)

- After using this tool, your students are ready to extend their knowledge by continuing on in a computer science curriculum. (56.5%)
- Using this tool eases and promotes the teaching of programming. (52.2%)

In ranking the top three characteristics dealing with auxiliary material and support that are perceived to be most significant, the total survey responses (71% response rate), the users (74% response rate) and the non-users (81% response rate) responded similarly. Table 18 shows these rankings.

Rank	Total	Users	Non-Users
1	This tool can be used independent of the textbook chosen	This tool can be used independent of the textbook chosen	Students enjoy using this tool
2	Students enjoy using this tool	Using this tool eases and promotes the learning of programming	Materials about this tool can be found through on-line searches
3	Using this tool eases and promotes the learning of programming	Students enjoy using this tool	This tool can be used independent of the textbook chosen

Table 18: Characteristics dealing with auxiliary materials and support that are perceived to contribute to the effectiveness of the tool as ranked by total survey responses, the responses of the users, and the responses of the non-users

The rankings of the top three characteristics dealing with auxiliary material and support that are perceived to be most significant for the effectiveness of this tool show that the male responses (94% response rate) and the female responses (79% response rate) were almost identical to the total survey responses. The only difference is the third ranked characteristic of the male responses. Table 19 shows these rankings.

The responses of the college teachers (91% response rate) and the responses of the secondary school teachers (85% repoonse rate) agreed with the rankings of the total survey reponses for the characteristics dealing with auxiliary materials and support that contribute to the effectiveness of the tool.

Rank	Total	Males	Females
1	This tool can be used independent of the textbook chosen	This tool can be used independent of the textbook chosen	This tool can be used independent of the textbook chosen
2	Students enjoy using this tool	Students enjoy using this tool	Students enjoy using this tool
3	Using this tool eases and promotes the learning of programming	Using this tool eases and promotes the teaching of programming	Using this tool eases and promotes the learning of programming

Table 19: Characteristics dealing with auxiliary materials and support that are perceived to contribute to the effectiveness of the tool as ranked by total survey responses, the male responses, and the female responses

23% of the total survey responses, 22% of the responses of the users, and 27% of the responses of the non-users identifed characteristics dealing with auxiliary material and support that are perceived to be inadequately supported by the tool. The top three of these characteristics (in priority order) are listed in Table 20.

Rank	Total	Users	Non-Users
1	Instructor resources (PP presentations, sample problems,labs, and syllabus) are provided by authors or community	Textbooks that incorporate this tool are available	Instructor resources (PP presentations, sample problems, labs, and syllabus) are provided by authors or community
2	Textbooks that incorporate this tool are available	A mechanism for the sharing of materials is provided by authors or community	Textbooks that incorporate this tool are available
3	A mechanism for the sharing of materials is provided by authors or community	Instructor resources (PP presentations, sample problems, labs, and syllabus) are provided by authors or community	Technical support is provided by the authors or the community

Table 20: Characteristics relating to auxiliary material and support, in priority order, that are perceived to be inadequately supported by the tool as ranked by total survey responses, the responses of the users, and the responses of the non-users

Table 21 reports the top three characteristics relating to auxiliary material and support that are not adequately supported by the tool. The rankings of the total survey responses

Rank	Total	Males	Females
1	Instructor resources (PP presentations, sample problems, labs, and syllabus) are provided by authors or community	Textbooks that incorporate this tool are available	Instructor resources (PP presentations, sample problems, labs, and syllabus) are provided by authors or community
2	Textbooks that incorporate this tool are available	Instructor resources (PP presentations, sample problems, labs, and syllabus) are provided by authors or community	Tutorials on using the tool are provided by authors or community
3	A mechanism for the sharing of materials is provided by authors or community	A mechanism for the sharing of materials is provided by authors or community	A mechanism for the sharing of materials is provided by authors or community

Table 21: Characteristics relating to auxiliary material and support, in priority order, that are perceived to be inadequately supported by the tool as ranked by total survey responses, the male responses, and the female responses

are compared to the male responses (30% response rate) and the female responses (30% response rate). Table 22 compares the the total survey responses to the responses of the college teachers (22.8% response rate) and the secondary school teachers (30%).

7.2.3.6 Negative Aspects of the Tool that may Hinder Teaching or Learning

This section queried teachers about characteristics of the tool that are perceived to hinder the learning or the teaching of programming. The first two questions provided a response choice of "no significant negatives." The majority of all groups indicated that the tool being evaluated had no significant negatives related to the tool environment, errors, or support.

Characteristics relating to the tool environment that may hinder teaching and learning

The first question in this section of the survey asked teachers to indicate the characteristics relating to the tool environment that they perceive to either hinder teaching or hinder the students' learning.

- 71.4% of the total survey responses included an answer to this question with 54.4% of those indicating that the tool had no significant negatives.

Rank	Total	College Teachers	Secondary School Teachers
1	Instructor resources (PP presentations, sample problems, labs, and syllabus) are provided by authors or community	Instructor resources (PP presentations, sample problems, labs, and syllabus) are provided by authors or community	Textbooks that incorporate this tool are available
2	Textbooks that incorporate this tool are available	A mechanism for the sharing of materials is provided by authors or community	Instructor resources (PP presentations, sample problems, labs, and syllabus) are provided by authors or community
3	A mechanism for the sharing of materials is provided by authors or community	Textbooks that incorporate this tool are available	A mechanism for the sharing of materials is provided by authors or community

Table 22: Characteristics relating to auxiliary material and support, in priority order, that are perceived to be inadequately supported by the tool as ranked by total survey responses, the responses of the college teachers, and the responses of the secondary school teachers

- 89% of the male responses included an answer to this question with 49.4% of those indicating that the tool had no significant negatives.
- 84% of the female responses included an answer to this question with 60.3% of those indicating that the tool had no significant negatives.
- 85% of the secondary school teachers' responses included an answer to this question with 55.6% of those indicating that the tool had no significant negatives.
- 87.7% of the college teachers included an answer to this question with 50% of those indicating that the tool had no significant negatives.
- 70% of the users included an answer to this question with 60.9% of those indicating that the tool had no significant negatives.
- 83.8% of the non-users included an answer to this question with 25.8% of those indicating that the tool had no significant negatives.

Characteristics relating to errors and support that may hinder teaching and learning

The second question in this section of the survey asked teachers to indicate the characteristics relating to errors and support perceived to either hinder teaching or hinder learning.

- 74% of the total survey respondents included an answer to this question with 56% of those indicating that the tool had no significant negatives.
- 92% of the male responses included an answer to this question with 53.3% of those indicating that the tool had no significant negatives.
- 88% of the female responses included an answer to this question with 62% of those indicating that the tool had no significant negatives.
- 88% of the responses of the secondary school teachers included an answer to this question with 55.4% of those indicating that the tool had no significant negatives.
- 95% of the responses of the college teachers included an answer to this question with 59.3% of those indicating that the tool had no significant negatives.
- 73% of the users' responses included an answer to this question with 61.5% of those indicating that the tool had no significant negatives.
- 81% of the non-users' responses included an answer to this question with 33.3% of those indicating that the tool had no significant negatives.

Although the majority of the responses indicated that the tool being evaluated had no characteristic that significantly hindered teaching or learning, the one characteristic that ranked in the top of those perceived as negative by all groups was:

- cryptic error messages (percentages ranging from 6.7% - 17.8%).

7.2.4 Aggregation: Summary of the Individual Tool Categories

This study is designed to determine the characteristics common to pedagogical tools that are perceived by teachers to be successful. Do successful tools share common characteristics? What are the characteristics that teachers perceive contribute to the

effectiveness of a successful pedagogical tool? In order to better analyze the perceptions of teachers regarding the characteristics of successful pedagogical tools used in introductory programming courses, this section investigates the responses of the aggregation of tool categories. The tool categories are defined in Section 3.1 of this dissertation. The number of survey responses and the tools included in each category are identified below:

- Microwords (78): Alice, Greenfoot, GridWorld, Jeroo, and Karel J. Robot
- Libraries (6): ObjectDraw and MediaComputation
- IDEs (119): BlueJ, DrJava, Dr Scheme, jGRASP, Netbeans, Eclipse, JCreator, and Greenfoot
- Visualizations (14): jGRASP and RAPTOR
- Robotics (1): Lego Mindstorms
- Games (2): Game Maker

Tools not mentioned in the above categories were not evaluated by any respondents and therefore not included in the results and analysis of this study and because of the low response rates for Robotics(1) and Game Maker(2), these two categories are not included in the aggregate summary of the results.

7.2.4.1 General Information About Choosing to Use the Tool

This section of the survey identified the reasons that teachers initially choose to use a pedagogical tool in an introductory programming class. Table 23 lists the reasons for initially choosing to use the tool. The majority choice responses (choices that had more than 50%) in each of the four tool categories are included in the table.

When asked to identify the one reason that was the most influential, responses in all four tool categories identified this reason to be:

- to introduce programming in a more enjoyable way

Microworlds 99% response rate	Libraries 100% response rate	IDEs 99% response rate	Visualization Tools 100% response rate
to introduce programming in a more enjoyable way (89.6%)	to introduce programming in a more enjoyable way (100%)	recommended to me by other computer science educators (63.36%)	to introduce programming in a more enjoyable way (57.1%)
to attract a diverse group of students (66.2%)	recommended to me by other computer science educators (66.7%)	to introduce programming in a more enjoyable way (50%)	task-technology fit (57.1%)
recommended to me by other computer science educators (62.3%)			

Table 23: Reasons for initially choosing to use this tool in an introductory programming course chosen by a majority of the responses in each tool category

7.2.4.2 Overall Tool Characteristics

The characteristics in this category focus on the user interface of the tool and student interaction.

Characteristics of the tool that are perceived to contribute to the tool's effectiveness

The characteristics perceived to contribute to the effectiveness of the tool by more than 50% of the responses in each tool category are displayed in Table 24. Table 25 lists, in order of priority, the rankings of the top three characteristics perceived to contribute to the effectiveness of the tool in an introductory programming class. The two characteristics ranked second for Visualization Tools had equivalent Borda numbers.

Characteristics not adequately supported by the tool (if any such characteristics are perceived to exist)

Also of interest is the teachers' perceptions of the characteristics that are not adequately supported by the tool. When considering the results by category, there were no responses to this question by those evaluating Libraries and only two responses from those evaluating Visualization Tools. Therefore, those tool categories are not included in Table

Microworlds 91% response rate	Libraries 100% response rate	IDEs 90% response rate	Visualization Tools 93% response rate
supports an interactive environment (85.9%)	improves first-time programmer's experience (100%)	has a learning curve that is not too steep (74.8%)	has a learning curve that is not too steep (100%)
improves first-time programmer's experience (84.5%)	supports an interactive environment (83.3%)	improves first-time programmer's experience (72%)	has an intuitive interface (84.6%)
includes graphical components (84.5%)	is flexible (allows use at many levels of learning) (83.3%)	supports an interactive environment (65.4%)	supports an interactive environment(84.6%)
has a learning curve that is not too steep (74.6%)	embodies ideas that support core programming concepts (83.3%)	has an intuitive interface (65.4%)	improves first-time programmer's experience (84.6%)
embodies ideas that support core programming concepts (69%)	includes graphical components (83.3%)	is flexible (allows use at many levels of learning) (58.9%)	is flexible (allows use at many levels of learning) (76.9%)
supports a concepts-first approach (67.6%)			
has an intuitive interface (63.4%)			
supports a consistent metaphor (59.2%)			
is flexible (allows use at many levels of learning)(59.2%)			

Table 24: Characteristics relating to user interface and student interaction perceived to contribute to the effectiveness of the tool by more than 50% of the responses in each tool category

Rank	Microworlds 87% response rate	Libraries 100% response rate	IDEs 88% response rate	Visualization Tools 93% response rate
1	improves first-time programmer's experience	improves first-time programmers experience	has a learning curve that is not steep	has a learning curve that is not steep
2	includes graphical components	includes graphical components	improves first-time programmer's experience	has an intuitive interface improves first-time programmer's experience
3	supports a concepts-first approach	embodies ideas that support core programming concepts	has an intuitive interface	

Table 25: Top three ranked characteristics, in priority order, that are perceived to contribute to the tool's effectiveness as determined by responses for each tool category

26, which identifies, in priority order, the top three characteristics that are not adequately supported by the tool.

7.2.4.3 Tool Characteristics Focusing on Programming Environment, Testing, Debugging, and Interaction

This section of the survey queried teachers about characteristics pertaining to the programming environment, debugging, testing, and interaction. The results are displayed as an aggregation of the separate tool category results.

Characteristics related to the programming environment that contribute to the effectiveness of this tool

The first question in this section focused on the programming environment characteristics perceived to contribute to the effectiveness of the tool. Table 27 identifies the characteristics chosen in more than 50% of the responses in three of the four tool categories. For libraries (100% response rate), no characteristic had a majority of responses and therefore is not included in the table.

Rank	Microworlds 39.7% response rate	IDEs 34% response rate
1	allows for easy transition to a more robust programming environment	allows for easy transition to a more robust programming environment
2	supports abstraction	includes graphical components
3	is flexible(allows use at many levels of learning)	improves a first-time programmer's experience

Table 26: Top three characteristics, in priority order, that are perceived to be inadequately supported by the tool as determined by the survey responses evaluating Microworlds and IDEs

Microworlds 87% response rate	IDEs 85% response rate	Visualization Tools 93% response rate
supports the visualization of the program state (64.7%)	simplifies the mechanics of programming (55.4%)	simplifies the mechanics of programming (61.5%)
supports visualization of OO concepts (58.8%)	provides multiple views at once (55.4%)	supports visualization of program code (61.5%)
supports visualization of state changes (54.4%)	supports visualization of OO concepts (50.5%)	supports visualization of program state (53.8%)
provides multiple views at once (54.4%)		provides multiple views at once (53.8%)
simplifies the mechanics of programming (52.9%)		

Table 27: Characteristics related to programming environment perceived to contribute to the effectiveness of the tool by more than 50% of those repsonding within each tool category

Characteristics that relate to testing, debugging, and interaction that contribute to the effectiveness of this tool

This question focusing on debugging, testing, and student interaction. Table 28 presents

the majority choices of the responses in three of the four tool categories. For libraries (100% response rate), no characteristic had a majority of responses and therefore is not included in the table.

Microworlds 78% response rate	IDEs 85% response rate	Visualization Tools 93% response rate
prevents syntax errors (52.5%)	supports debugging (74.3%)	supports step-by-step evaluation of single programming statements (84.6%)
supports an easy way of including event-driven programming (50.8%)	supports comments and documentation (70.3%)	supports debugging (84.6%)
	supports step-by-step evaluation of single programming statements (60.4%)	gives immediate feedback about errors (53.8%)
	gives immediate feedback about errors (57.4%)	supports comments and documentation (53.8%)
	supports developing and testing of individual components (55.4%)	

Table 28: Characteristics related to testing, debugging, and interaction perceived to contribute to the effectiveness of the tool by the majority of the responses in each tool category

The ranking of the top three characteristics, in order of priority, pertaining to the programming environment, debugging, testing, and interaction that are perceived to contribute to the effectiveness of the tool are presented in Table 29. These results are listed by category. The four characteristics listed as top ranked in the Libraries category had equal Borda numbers. In the Visualization category, the characteristics ranked second had equal Borda numbers.

Characteristics related to programming environment, debugging, testing, and interaction that are not adequately supported by the tool (if any such characteristics exist)

Table 30 identifies the characteristics that are perceived to be inadequately supported by

Rank	Microworlds 71% response rate	Libraries 67% response rate	IDEs 82% response rate	Visualization Tools 86% response rate
1	simplifies the mechanics of programming	supports the visualization of program state	supports visualization of OO concepts	supports step-by-step evaluation of single programming statements
		provides a media rich programming environment		
		supports developing and testing of individual components		
		supports an easy way of including event-driven programming		
2	supports the visualization of program state		supports debugging	simplifies the mechanics of programming
				supports debugging
3	supports the visualization of state changes		simplifies the mechanics of programming	

Table 29: Top three ranked characteristics, in priority order, that relate to the programming environment, testing, debugging, and interaction that are perceived to contribute to the effectiveness of this tool as determined by responses for each tool category

the tool. The top three characteristics (in priority order) ranked in the responses for three of the four tool categories are included in the table. The Libraries category had only two responses for this question and therefore those results are not included in the table. The Visualization Tools category had only a 35% response rate. Although the Borda numbers clearly indicate a top ranked characteristic in this category, the second rank was not so clear having a five-way tie with only one response for each of the five characteristics. Only the top ranked characteristic is included in Table 30 for Visualization Tools.

Rank	Microworlds 41% response rate	IDEs 49% response rate	Visualization Tools 36% response rate
1	provides meaningful error messages	prevents syntax errors	supports and easy way of including event-driven programming
2	supports step-by-step evaluation of single programming statements	supports visual representation of general programming concepts	
3	supports introduction of data structures	gives immediate feedback about errors	

Table 30: Characteristics, in priority order, that are perceived to be inadequately supported by the tool as determined by responses in each tool category

7.2.4.4 Tool Characteristics Contributing to the Learning of and the Teaching of Programming

This section of the survey queried teachers about characteristics pertaining to the learning and the teaching of programming.

Table 31 identifies the characteristics that are perceived to contribute to the effectiveness of the tool as they relate to the students' learning to program by a majority of responses in each of the four categories.

table31

Table 32 identifies the characteristics that are perceived to contribute to the effectiveness of the tool as they relate to the teaching of programming in introductory programming classes by a majority of responses in each of the four categories.

The rankings of the top three characteristics related to the learning and the teaching of programming as determined by the responses in each of the four tool categories are presented in Table 33.

Microworlds 82% response rate	Libraries 100% response rate	IDEs 77% response rate	Visualization Tools 71% response rate
supports students' active engagement in learning activities (93.8%)	supports students' active engagement in learning activities (100%)	supports students' active engagement in learning activities (60.9%)	supports students' active engagement in learning activities (60%)
engages students of different ability levels (81.3%)	engages students of different ability levels (83.3%)	helps understanding of program execution (57.6%)	helps understanding of program execution (60%)
encourages learning through discovery (79.7%)	supports the understanding of abstract and complex concepts (66.7%)	supports the understanding of abstract and complex concepts (52.2%)	
helps understanding of program execution (65.5%)	allows students to work at levels of their choice (66.7%)	engages students of different ability levels (51.1%)	
supports the understanding of abstract and complex concepts (64.1%)	helps understanding of program execution (66.7%)		
embodies ideas that support core programming concepts (69%)			
allows students to work at levels of their choice (60.%)			

Table 31: The characteristics that are perceived to contribute to student learning identified by more than 50% of the responses in each tool category

Microworlds 79% response rate	Libraries 100% response rate	IDEs 81% response rate	Visualization Tools 79% response rate
is a good tool for classroom demonstrations (91.9%)	is a good tool for classroom demonstrations (83.3%)	can be used for the duration of the course (87.5%)	is a good tool for classroom demonstrations (90.9%)
	can be used for the duration of the course (83.3%)	is a good tool for classroom demonstrations (77.1%)	can be used for the duration of the course (90.9%)

Table 32: The characteristics related to the teaching of programming that are perceived to contribute to effective teaching identified by more than 50% of the responses in each tool category

Rank	Microworlds 73% response rate	Libraries 83% response rate	IDEs 76% response rate	Visualization Tools 79% response rate
1	supports students' active engagement in learning activities	supports students' active engagement in learning activities	can be used for the duration of the course	is a good tool for classroom demonstrations
2	engages students of different ability levels	can be used for the duration of the course	is a good tool for classroom demonstrations	can be used for the duration of the course
3	encourages learning through discovery	allows students to work at levels of their own choice	supports the understanding of abstract and complex concepts	helps understanding of program execution

Table 33: Characteristics, in priority order, that are perceived to contribute to the effectiveness of the tool as related to the learning of and teaching of programming as determined by responses for each tool category

7.2.4.5 Materials, Support, and Teacher Perception of Tool Use

Table 34 identifies those characteristics related to auxiliary materials and support that are perceived to contribute to the effectiveness of the tool. The table identifies the characteristics chosen by the majority of responses in each tool category.

Microworlds 81% response rate	Libraries 100% response rate	IDEs 80% response rate	Visualization Tools 79% response rate
Materials about this tool can be found through on-line searches (82.5%)	Technical support is provided by the authors or the community (100%)	This tool can be used independent of the textbook chosen (90.5%)	This tool can be used independent of the textbook chosen (100%)
This tool can be used independent of the textbook chosen (81.0%)	Instructor resources are provided by the authors or the community (100%)	Materials about this tool can be found through on-line searches (68.4%)	Tutorials on using the tool are provided by authors or the community (63.6%)
Technical support is provided by authors or the community (68.3%)	This tool can be used independent of the textbook chosen (100%)	Tutorials on using the tool are provided by the authors or the community (67.4%)	
Instructor resources are provided by the authors or the community (68.3%)	Textbooks that incorporate this tool are available (83.3%)	Technical support is provided by the authors or the community (62.1%)	
Tutorials on using the tool are provided by authors or community (66.7%)	Materials about this tool can be found through on-line searches (66.7%)	Textbooks that incorporate this tool are available (50.5%)	
textbooks that incorporate this tool are available (61.9%)			

Table 34: The characteristics related to auxiliary materials and support that are perceived to contribute to effective teaching identified by more than 50% of the responses in each tool category

The overall perception of tool use in an introductory programming class is identified in Table 35. The characteristics included in the table are those chosen by more than 50% of the responses in each tool catagory.

The top three characteristics relating to auxiliary materials and support and overall

Microworlds 74% response rate rate	Libraries 100% response rate rate	IDEs 75% response rate rate	Visualization Tools 71% response rate rate
Students enjoy using this tool (93.1%)	The initial experience with this tool psoitively affects subsequent programming experiences (100%)	After using this tool students are ready to extend their knowledge by continuing on in a CS curriculum (79.8%)	Using this tool eases and promotes the learning of programming (90%)
Using this tool eases and promotes the teaching of programming (81.0%)	Using this tool eases and promotes the learning of programming (100%)	using this tool eases and promotes the learning of programming (76.4%)	Using this tool eases and promotes the teaching of programming (70%)
Using this tool eases and promotes the learning of programming (79.3%)	Students enjoy using this tool (100%)	Using this tool eases and promotes the teaching of programming (68.5%)	Students enjoy using this tool (70%)
The initial experience with this tool positively affects subsequent programming experiences (70.7%)	Using this tool eases and promotes the teaching of programming (83.3%)	Students enjoy using this tool (67.4%)	
After using this tool students are ready to extend their knowledge by continuing on in a CS curriculum (65.5%)		The initial experience with this tool positively affects subsequent programming experiences (60.7%)	

Table 35: The characteristics related to the overall perception of the tool identified by more than 50% of the responses in each tool category

perception of the tool are presented in Table 36.

When asked about characteristics that are desired but are not adequately supported by the tool, the response rates for all four categories were very low. There were no responses for Libraries, a 25.6% response rate for Microworlds, a 21% response rate for IDEs, and a 21.4% response rate for Visualizations. Table 37 lists, in priority order, the characteristics that are perceived to be inadequately supported for Microworlds, IDEs, and Visualiztion Tools based on these small response rates.

Rank	Microworlds 69% response rate	Libraries 83% response rate	IDEs 73% response rate	Visualization Tools 71% response rate
1	Students enjoy using this tool	Students enjoy using this tool	This tool can be used independent of the textbook chosen	This tool can be used independent of the textbook chosen
2	This tool can be used independent of the textbook chosen	This tool can be used independent of the textbook chosen	Using this tool eases and promotes the learning of programming	Using this tool eases and promotes the teaching of programming
3	Instructor resources are provided by authors or the community	Using this tool contributes positively to student retention in course	Technical support is provided by authors or community	Using this tool eases and promotes the learning of programming

Table 36: Top three characteristics, in priority order, that are perceived to contribute to the effectiveness of the tool as related to auxiliary materials, support, and overall perceptions of the tool as determined by responses for each tool category

Rank	Microworlds 26% response rate	IDEs 21% response rate	Visualization Tools 21% response rate
1	A mechanism for sharing materials is provided by authors or community	Textbooks that incorporate this tool are available	This tool can be used independent of the textbook chosen
2	Instructor resources are provided by authors or community	Instructor resources are provided by authors or community	A mechanism for sharing materials is provided by authors or community
3	Textbooks that incorporate this tool are available	Tutorials on using the tool are provided by authors or community	Tutorials on using the tool are provided by authors or community

Table 37: Characteristics related to auxiliary materials and support, listed in priority order, that are perceived to be inadequately supported by the tool as determined by responses for three tool categories

7.2.4.6 Negative Aspects of the Tool that may Hinder Teaching or Learning

This section of the survey queried teachers about the negative aspects of the tool environment in that they either hinder teaching or hinder the students' learning. The survey included a "no significant negatives" option. The majority of responses chose this option:

- Microworlds: 76.9% of those evaluating Microworlds answered this question; 45% of those responding indicated that the tool had no significant negatives.
- Libraries: 100% of those evaluating Libraries answered this question; 100% of those responding indicated that the tool had no significant negatives.
- IDEs: 67.2% of those evaluating IDEs answered this question; 61.3% of those responding indicated that the tool had no significant negatives.
- Visualization Tools: 64% of those evaluating Visualization Tools answered this question; 88.9% of those responding indicated that the tool had no significant negatives.

When asked to identify the characteristics that hindered teaching or learning, no characteristic was identified by a majority of responses in any tool category. No characteristics were identified for Libraries and only one response identified a characteristic for Visualization Tools. Table 38 identifies the three characteristics related to the programming environment that are perceived to hinder teaching or learning. The characteristics identified in the table are those with the largest percent of responses in each category.

The question pertaining to the characteristics related to errors and support that are perceived to hinder teaching or learning also had large percentages choosing "no significant negatives."

- Microworlds: 71.7% of those evaluating Microworlds answered this question; 51.8% of those responses indicated that the tool had no significant negatives.

Microworlds 77% response rate	IDEs 67% response rate
The transition to real world programming is difficult (25%)	Students have technical difficulties (20%)
The tool is too restrictive to use for the duration of the course (23.3%)	The learning curve is too steep (15%)
Good OO style is distorted by pragmatics and limitations of of this tool (15%)	The tool is too big for introductory classes (10%)

Table 38: Characteristics related to the tool environment that are perceived to hinder teaching or learning by those survey responses evaluating Microworlds and IDEs

- Libraries: 100% of those evaluating Libraries answered the question; 66.7% of those responses indicated that the tool had no significant negatives.
- IDEs: 73.9% of those evaluating IDEs answered this question; 61.4% of those responses indicated that the tool had no significant negatives.
- Visualization Tools: 78.5% of those evaluating Visualization tools answered this question; 90.9% of those responses indicated that the tool had no significant negatives.

The characteristics related to errors and support that were perceived to hinder teaching or learning for each tool category are listed in Table 39.

7.2.5 Training and Experience

The questions in this section of the survey focused on the training that teachers had in preparation for using the tool, the effectiveness of the tool in comparison to other tools they may have used in their class, the experience the teacher has in teaching the introductory programming course, and the experience the teacher has using the particular tool that is being evaluated. Both the cross-categories summary and the aggregation summary are reported in this section.

Microworlds 72% response rate	Libraries 100% response rate	IDEs 74% response rate	Visualization Tools 79% response rate
cryptic error messages (17.9%)	cryptic error messages (16.7%)	cryptic error messaages (13.6%)	cryptic error messages (9.1%)
no easy way to debug (16.1%)	does not prevent syntax errors (16.7%)	does not prevent syntax errors (12.5%)	
technical difficulties with installation (10.7%)	no easy way to debug (16.7 %)	technical difficulties with installation (6.8%)	
no technical support (10.7%)	technical difficulties with installation (16.7%)		

Table 39: The characteristics related to errors and support that are perceived to hinder teaching or learning as determined by responses for each tool category

Amount of training

The survey collected information on teacher's training in the use of this tool before the tool was introduced in an introductory programming class. The results were similar for all groups across categories (users, non-users, male respondents, female respondents, college teachers, and secondary school teachers). More than 48% of the survey responses indicated having no training, knew nothing about tool, and learned by experimentation. Approximately 25% of the survey responses indicated observing a demonstration of the tool at a conference but had no other training. Another 32% - 40% indicated learning the tool through a tutorial provided with the tool or learning through a user's guide or training manual that accompanied the tool.

When looking at the individual tool categories, results were similar and are presented in Table 40.

To investigate the need for training and the training effectiveness, and to query teachers about the overall perception of tool use in an introductory programming class, respondents were asked to answer a series of questions in which a 5-point Likert scale was used as the research instrument. The questions provide a set of response items. For each item, the

Training	Microworlds 83% response rate	Libraries 100% response rate	IDEs 79% response rate	Visualization Tools 79% response rate
no training	43.0%	16.7%	52.2%	63.6%
demonstration	29.2%	33.3%	24.5%	27.3%
workshop	31.0%	33.3%	22.4%	18.2%
manual or user's guide	55.4%	50.0%	28.8%	36.4%

Table 40: Type of training received in the use of the tool for each tool category

respondents were asked to choose from varying degrees of agreement on a scale ranging from *strongly agree* (1) to *strongly disagree* (5). This set of choices includes a neutral choice for those who are uncertain about their agreement with a particular statement.

The mean and standard deviation are generally used when reporting continuous data. Since Likert-scale data are of an ordinal nature, the median and mode may have more meaning in the Likert-scale analysis in this study than do the mean and standard deviation. Both sets of descriptive statistics (ordinal and continuous) are included here. Tables 41 and 42 report the frequency, mean, and mode for the Likert-scale questions used in this study. Table 41 displays these descriptive statistics for the total survey responses and Table 42 displays the same statistics for the responses of those presently using the tool being evaluated in an introductory course (eliminating the non-users). The means and standard deviations for the same groups (total survey responses and users) are reported in Tables 43 and 44 respectively. As with other questions in this survey, the Likert-scale items do not reflect measured student learning outcomes, but rather the teachers' beliefs and attitudes. The survey is intended to collect and record the teachers' perceptions of tool use in introductory programming classes.

Statement	N	1 Strongly Agree	2 Agree	3 Neither Agree nor Disagree	4 Disagree	5 Strongly Disagree	Median	Mode
The amount of training I received in the use of this tool was enough.	180	59	65	31	21	4	2	2
I will continue to use the tool in future courses.	182	93	61	11	10	7	2	2
I will recommend this tool to other computer science educators	182	92	67	13	4	6	1	1
Using this tool in my introductory programming class eases and promotes the LEARNING of programming concepts	181	71	76	22	5	7	2	2
Using this tool in my introductory programming class eases and promotes the TEACHING of programming concepts	180	64	87	20	4	5	2	2
Using this tool in my introductory programming class eases and promotes the LEARNING of OO concepts	180	58	65	33	16	8	2	2
Using this tool in my introductory programming class eases and promotes the TEACHING of OO concepts	178	48	74	36	12	8	2	2

Table 41: Descriptive statistics (frequency, median, mode) for the total survey responses

Statement	N	1 Strongly Agree	2 Agree	3 Neither Agree nor Disagree	4 Disagree	5 Strongly Disagree	Median	Mode
The amount of training I received in the use of this tool was enough.	149	53	52	26	15	3	2	1
I will continue to use the tool in future courses.	150	89	55	5	1	0	1	1
I will recommend this tool to other computer science educators	150	85	58	6	1	0	1	1
Using this tool in my introductory programming class eases and promotes the LEARNING of programming concepts	150	67	64	16	2	1	2	1
Using this tool in my introductory programming class eases and promotes the TEACHING of programming concepts	149	61	70	16	2	0	2	2
Using this tool in my introductory programming class eases and promotes the LEARNING of OO concepts	149	56	53	26	12	2	2	1
Using this tool in my introductory programming class eases and promotes the TEACHING of OO concepts	147	47	62	26	10	2	2	2

Table 42: Descriptive statistics (frequency, median, mode) for the responses of those presently using the tool

Statement	N	Minimum	Maximum	Mean	Standard Deviation
The amount of training I received in the use of this tool was enough.	180	1.00	5.00	2.1444	1.07341
I will continue to use the tool in future courses.	182	1.00	5.00	1.7747	1.04528
I will recommend this tool to other computer science educators	182	1.00	5.00	1.7088	0.93892
Using this tool in my introductory programming class eases and promotes the LEARNING of programming concepts	181	1.00	5.00	1.9006	0.98379
Using this tool in my introductory programming class eases and promotes the TEACHING of programming concepts	180	1.00	5.00	1.8833	.89239
Using this tool in my introductory programming class eases and promotes the LEARNING of OO concepts	180	1.00	5.00	2.1722	1.11280
Using this tool in my introductory programming class eases and promotes the TEACHING of OO concepts	178	1.00	5.00	2.2022	1.23222

Table 43: Descriptive statistics (mean and standard deviation) for the total survey responses

Statement	N	Minimum	Maximum	Mean	Standard Deviation
The amount of training I received in the use of this tool was enough.	149	1.00	5.00	2.0805	1.05598
I will continue to use the tool in future courses.	150	1.00	4.00	1.4533	0.59738
I will recommend this tool to other computer science educators	150	1.00	4.00	1.4867	0.61017
Using this tool in my introductory programming class eases and promotes the LEARNING of programming concepts	150	1.00	5.00	1.7067	0.76454
Using this tool in my introductory programming class eases and promotes the TEACHING of programming concepts	149	1.00	4.00	1.7248	0.70576
Using this tool in my introductory programming class eases and promotes the LEARNING of OO concepts	149	1.00	5.00	2.0000	1.00000
Using this tool in my introductory programming class eases and promotes the TEACHING of OO concepts	147	1.00	5.00	2.0340	0.94662

Table 44: Descriptive statistics (mean and standard deviation) for the responses of those presently using the tool

7.2.6 Course Description and Teacher Experience

This study investigates the use of pedagogical tools in an introductory programming course. Programming is introduced in varying degrees of intensity. Middle school students may be introduced to programming concepts using Scratch or Game Maker while college students may have a first-time programming experience with Alice or Java. The tools chosen are not grade-level tools. Scratch is used both in middle schools and in colleges; Java is used at the high school level and post-secondary levels. Educators charged with teaching introductory programming courses accept the challenge with diverse teaching backgrounds and varied knowledge about and experience with pedagogical tools.

7.2.6.1 Course Description

Introductory programming is taught at many educational levels in courses varying in length, rigor, and requirements. In secondary schools, the titles of the courses taught by the survey respondents include such names as "Computing in the Modern World", "Introduction to Java," and "Advanced Placement Computer Science." These courses vary in length from 30 hours to 270 hours. The average course length of 129.1 hours; The median length is 130 hours; The mode length is 180 hours. 96% of the respondents have a course length less than or equal to 180 hours and approximately 69% of the respondents' courses have a length between 100 and 180 hours.

At the college level, most titles reflect "Introduction to Object-Oriented Programming" or "Computer Science 1." The courses at the college level vary in length from 30 hours to 168 hours with an average course length of 62 hours, a median course length of 45 hours and a mode length of 45 hours. 90% of the respondents have a course length less than or equal to 80 hours and approximately 66% of the respondents have a course length betweeen 40 and 80 hours.

Introductory programming courses have different statuses in different educational environments. The survey asked teachers to identify their course by choosing from the following:

- Elective (55.9%): Elective courses are optional courses that a student chooses to take. These courses are not required to graduate or to fill any graduation requirement.
- General education requirement (8.4%): General education requirements describe core courses that all students must take in order to graduate. In some schools a programming course may satisfy a quantitaive requirement.
- Majors requirement (23.5%): At the college level, students generally declare a major. The introductory programming course may satisfy a requirement for the computer science (or other) major.
- Honors course (4.5%): At the secondary school level, some courses are designated as honors. This designation indicates that the course is of a higher level of difficulty. The grade in the honors course is usually weighted differently and can strengthen the student's chances for college admission.
- AP course (33.5%): At the secondary level, AP courses prepare the student to take the College Board's Advanced Placement Examination. Passing this exam may earn the student college credit. Somewhat equivalent to the AP course difficulty is the International Baccalaureate (IB) course. An AP or IB course usually carries at least the weight of an honors course.

This question was answered on 80% of the total survey responses. The percentages in parentheses above indicate the percentage of responses whose courses best fit that description. The choices are not mutually exclusive. For example, an AP Computer Science course may also be an elective course.

7.2.6.2 Teacher Demographics and Experience

The survey respondents in this study are not intended to be representative of the typical teachers of introductory programming classes. As explained in Section 3.6, the

respondents are likely to be at a higher level of professional involvement in computer science education than most teachers of introductory programming courses. 81% of those completing the survey chose to answer the demographic questions. Of these responses, 55.2% were male and 44.8% were female. For the secondary school responses, 48.8% were male and 51.2% were female. At the college level, 66.7% were male and 33.3% were female.

The survey responses identified 71.3% teaching at the secondary school level (1.7% middle school, 69.6% high school) and 32.4% teaching at the college level (29.3% at four-year institutions and 3.1% at two-year institutions). A few respondents were teaching at both college and secondary levels.

In response to teaching experience and experience using the pedagogical tool being evaluated, 82.6% of the responses of those teaching in the secondary schools have taught the introductory programming course five or more times with 43.1% of these teachers having used the evaluated tool five or more times. 15.9% of the responses indicate having used the tool only once.

At the college level, 89.1% of the responses indicate having taught the introductory programming course five or more times with 46.5% of them using the evaluated tool five or more times. Only 10.7% of the college responses indicate having used the tool only once.

7.3 Interviews with Teachers

The researcher recognizes the importance of multiple data sources. Data for this study was collected from three main sources: teacher survey, teacher interviews, and interviews with the tool developers. Each data source provides additional information about the tools as well as different perspectives on tool use in introductory programming courses and allows the researcher to explain more fully the common characteristics of an effective pedagogical tool and its use from more than one viewpoint, making use of the different data sources [111, 46]. This section reports on the information collected from the teacher interviews.

Standardized open-ended interviews were conducted with eleven teachers of introductory

programming. Of these eleven teachers, four were from the university, six were high school teachers (grades 9 - 12) and one was a middle school computing teacher. Appendix C contains the teacher interview questions. Summaries of the responses to these questions are reported here.

How does a teacher find appropriate tools to use?

All interview respondents identified the *computer science community* as the primary way to find appropriate tools to use in an introductory programming course. Individual teachers identified the community as:

- Special Interest Group on Computer Science Education mailing list (SIGCSE Announce)
- Advanced Placement Computer Science Electronic Discussion Group (AP CS EDG) Both SIGCSE Announce and AP CS EDG are electronic discussion groups that allow for communication via email with other members of the community.

Individual teachers also identified sources for information on pedagogical tools as:

- Searches on the Internet
- Conversations with other educators
- Professional development workshops or symposia

What influences a teacher to use one particular tool over another?

There were two major themes in the answers to this question:

- Teacher recommendation: based on the perceived credibility of the people on the mailing lists recommending the tool.
- Fits the need of the course. As one teacher describes, "...the tool replaces something for which it does better or it adds to what I already have in place in terms of instruction."

What are your reactions to: Teachers do not use tools in their class because a) they do not have time to spend learning the tool or b) they cannot afford the time to use the tool in their class because it would take time away from the curriculum they must cover.

Most interviewees respond to these statements "with sympathy and skepticism." They agree that there is truth to the statements but that time put into learning the tool will be worth it in the long run. Specific reactions from individual teachers illustrating this are:

- "If you find something that is going to replace what you are currently doing and make it better, then you are going to find the time to learn it. I think some tools just help you teach better, more effectively, and quicker when you use them and that's good."

- "If it's a good tool, it will end up making you time. At the end of the course, or the end of the year, or the end of the student's degree, you want them to have a certain set of skills or learning outcomes and if you are going to get them to have more of them or have them deeper, that's good. A good tool will help you with better learning outcomes or different learning outcomes or deeper learning outcomes. A good tool will help you give them all of those. That pretty much translates into saving you time, giving you more time to do more different things so they get those better learning outcomes."

- " it doesn't matter that you covered a lot of stuff if the students don't get it and so I argue back and when I am trying to help other professors and teachers, I try to show that; give them something that gets them deeper or further than they would have gone and then you can grow it from there with a constructivist kind of approach."

What are your reactions to: Many teachers using tools learn how to use them without any formal training.

Most of the interviewees agree that:

- Teachers, especially in the field of computer science, learn tools by "fiddling around" with them.

- Teacher experience helps in the learning process.
- Fiddling may allow the teacher to learn *how* the tool works but won't offer much in using the tool effectively within a pedagogical framework.

What do you believe is the best medium to train in the use of a tool?

The interviewees agree that different people learn in different ways. Most respondents mentioned two training mediums:

- Workshops (of any length): Face-to-face workshops allow for active engagement of the participants. The tool and its pedagogical strengths can be observed.
- Videos: Short video clips on the tool use and pedagogical strengths.

When queried about Web training, most of those interviewed would participate in a Web training event if it did not require a great deal of time.

Do you believe that you use the tool to its full potential?

All interview respondents agree that the tools used in their classes are not used to their full potential. Most interviewees use 50% - 75% of the tool's capabilities, choosing aspects of the tool based on their curriculum needs.

What do the authors of the tools need to know or to ask the teachers?

- Ask what works and what doesn't work.
- Ask what could be done better
- Find people to develop lessons and activities: out of the box activities that come with video or podcast, not just a text document.
- Provide pedagogical examples of how your tool is strong and provide the media that backs it up.

- The tool has to be plug and play with no hang-ups on installation.
- Provide tutorials for introductory students and make the tutorials easy to find.
- Ask about the type of students that will be using the tool and provide for student needs at various levels.
- Provide a mechanism to share resources.

How to you disseminate information about a tool that you believe in?

The interview respondents were in total agreement that the AP CS EDG was the best way to disseminate information about pedagogical tools to the computer science community. When reminded that not all computing teachers are members of the AP CS community, several other suggestions were offered:

- Computer Science & Information Technology (CS & IT) Symposium: provides professional development opportunities for high school and middle school computer science and computer applications teachers who need practical, relevant information to help them prepare their students for the future [56]. Approximately 200 teachers attend.
- SIGCSE Symposium: The SIGCSE Technical Symposium addresses problems common among educators working to develop, implement and/or evaluate computing programs, curricula, and courses. The symposium provides a forum for sharing new ideas for syllabi, laboratories, and other elements of teaching and pedagogy, at all levels of instruction [2]. Approximately 1150 computer science educators attend with about 50 of those registering as high school teachers.
- Consortium for Computing Sciences in Colleges (CCSC): Conferences held in ten regions of the country primarily for two- and four-year college computer science educators. Special sessions for K-12 teachers are encouraged and K-12 teachers are invited to attend [72].

- Computer Science Teachers Association (CSTA) Web Repository: A searchable database of instructional materials, lesson plans, and other resources that have never before been collected in one place for use by all CS teachers [56].
- National Education Computing Conference (NECC): Boasts to be the world's most comprehensive education technology event. The NECC Program features a vast array of professional learning and collaborative networking opportunities [73].
- Individual state computing conferences
- Computing contests
- Blogs

Do you use any tool for demonstration purposes only?

Although none of the interviewees use pedagogical tools only for demonstration purposes in introductory programming courses, visualization tools are used by a few to display data structures and other concepts in more advanced courses.

Would your teaching change if these pedagogical tools were taken away from you?

All respondents agreed that the way they teach would change significantly if the tools were removed from their teaching repertoire. All would adapt by recreating most of their lessons and replacing some aspects of tool use with "manipulatives" and role playing. Others agreed that their lessons would be somewhat dull.

What tools do we need that are not out there?

Some suggestions the interviewees had were:

- A better platform for Web IDE
- Better visualization tools (working at different levels and in different languages)
- A system of small interactive units to teach introductory concepts

- An environment that includes Karel
- A cognitive tutor environment that remembers student interactions and monitors student progress

Are there any tools that you consciously choose not to use"?

A majority of the respondents have consciously chosen not to use certain tools in their introductory courses. Many have switched from one tool to another. The reasons vary.

- "I switched to Alice because it's newer and different."
- "I did use Alice but the time spent on it didn't give a good return."
- "I switched to Alice because I wanted to hand kids something that in 15 minutes they could see an ice skater dancing on the screen."
- "I wanted to use Karel but I couldn't get it to work easily."
- "I won't use Game Maker again until I find a better way to teach it. Maybe with the scripting."
- "I dropped Robo Code. It's too hard to ramp up on."
- "I don't use Scratch because I don't know what it can be used for."
- "I don't use Eclipse because it's too complicated."
- " I am switching from Alice to Scratch. I'll be interested in the next Alice version but I think Scratch will be better right now for what I want."
- "I don't use algorithm visualization tools. It seems that they are overwhelming. It would be nice to have something that would be easy for students without the overhead."

Some respondents chose not to use the tool because it was not appropriate for the level or content of their course. Table 3, Section 7.2, contains a listing of tools that the survey

respondents either use or consciously choose not to use in their introductory programming course. The *not-using* column was the focus of this question.

If you could choose only one tool to keep using, what would it be?

There was no consensus among the respondents. Of the eleven teachers responding, there were eight different answers. They included: BlueJ, JCreator, DrScheme, DrJava, Flash, Greenfoot, Alice, and Eclipse. However, most respondents indicated that the reason for keeping this one tool was that the tool allowed for the coverage of the topics most critical in the course and the tool can be used for the duration of the course.

7.4 Interviews with Tool Developers

As mentioned throughout this dissertation, the data for this study was collected from three main sources: teacher surveys, interviews with teachers, and interviews with the those that design and develop the tools. Interviews with the developers of the tool provide a different perspective than do the interviews with the teachers (users of the tool).

The third and final summary of data collection in this study involves interviews with the tool developers. Semi-structured interviews were conducted with six individuals, each of whom is a tool developer or a member of the tool design and development team for one of the following pedagogical tools: Alice, BlueJ/Greenfoot, Java Task Force Library, Jeroo, Karel J. Robot, and ObjectDraw. Appendix C contains the interview questions. Summaries of the responses to these questions are reported here.

- When asked, "What influenced you to create your tool?"
  the developers identified three major reasons:
  - to provide a way of teaching object-oriented programming well from the beginning
  - to modernize the course and make it more appropriate and enjoyable for today's students

  - to include, within the teaching environment, a form of visualization or graphics so the student could actually author something and then watch how the program executed.

  Several interviewees credited Rich Pattis, author of the original *Karel the Robot: A Gentle Introduction to the Art of Programming*, as the impetus for creating pedagogical tools or environments for the teaching of programming concepts.

- There was consensus that the tool created was created for its pedagogical importance and not for the tool's technological value. Pedagogically, developers agreed that visual representation of the student's coded work and communicating the deeper, more complicated programming concepts relatively easily and quickly were goals in the tool development. Specific goals of some of the tool developers illustrating this include:

  - "We have the ability to go stepwise and go at various speeds so students can see the effect and understand the effect of their code."
  - "The student could actually author something and then could watch how the program executed...would fit much stronger into helping the student to learn."
  - "...we remove the tedium but keep the essential ideas."
  - "the student can relatively easily and relatively quickly get to deep ideas about polymorphism..."
  - "from the beginning they (the students) would be thinking in the right way, they'd be developing solutions in the right way, so they didn't have to completely relearn everything when they learned how to do it the real way."

  Technologically, Alice was created at the time when 3-D graphics were first available on the personal computer and so Alice did make a technological impact on the pedagogical tool development.

- When asked about particular strengths and weaknesses of the tool, all tool developers noted that the tool supports sound pedagogical goals providing a

framework for talking about deep programming concepts without the overhead of dealing with language details. The different weaknesses mentioned by the tool developers included:

  - Some tool developers aren't thrilled with the code behind the scenes but if it's not open source, no one sees the code.
  - There are difficulties dealing with input and output.
  - The two-dimensional environment is not as attractive as 3D.
  - The tool is a bit old and needs revision.
  - The transition to "real world" may be difficult.

- The pedagogical tools investigated by this study are developed for teachers to use in introductory programming courses. When asked if the developers seek feedback and reactions about their tools, all tool developers are *receptive* to receiving feedback from the teachers using their tools and respond to questions and concerns that teachers pose through email. Not all tool developers actively seek feedback. Some tool developers run workshops at which time they request reactions and feedback. All of the tool developers interviewed provide a mechanism on the tool web site for receiving feedback.

- A major source for learning about pedagogical tools is the computer science education communities. Two active communities links are SIGCSE Announce and the AP CS EDG. The SIGCSE mailing list serves mostly college computer science educators while the AP CS list reaches primarily the high school computer science teachers. Though not all interviewees are active participants on these lists, most are "listeners" or have others on the tool development team who are active. All of the interviewees are members of the SIGCSE community and all but one of the interviewees belong to the AP CS list.

- Proliferating information about the tool is important if the tool is to be accepted and used in the community of computer science educators. Most of the developers

advertise their tool by word of mouth. Some developers offer workshops at the SIGCSE Symposium. But, as one developer states: "High school teachers are not well-organized above the local level and so they are harder to reach." The CSTA CS & IT Symposium, primarily for K-12 computing educators, reaches only about 200 teachers. The SIGCSE symposium reaches approximately 1150 computer science educators of which about 50 are registered as high school teachers. Other attempts to disseminate information about the tool have included: hosting competitions that use the tool, distributing trinkets or wearing T-shirt that advertise the tool. Most admit that advertising is easy but getting people to find the advertisement is challenging. The primary means is electronic communication through the mailing lists or tool web-sites.

- Most of the tool developers interviewed offer some form of teacher resources available free on-line.

The strengths of this research are based on the gathering of data from three sources: a survey to teachers, interviews with teachers, and interviews with tool developers. This chapter has presented the results of each of the three data collections separately. Chapter 8 will classify, categorize, and summarize this data providing an analysis of the results in answering the research questions that were posed in Chapter 3.

Chapter 8

Analysis of Results

8.1 Introduction

The data for this study was collected from three main sources: teacher surveys, interviews with teachers, and interviews with the tool developers. The interviews give greater depth into the choice and use of pedagogical tools in the introductory programming class and the survey provides greater breadth of the tool use. Interviews with the developers of the tools provide a different perspective than do the interviews with the teachers (users of the tool). The three data sources provide a methodological triangulation for the results of this study. "Adding one layer of data to another builds a confirmatory edifice [58]." Teddie and Tashakkori define triangulation as the combinations and comparisons of multiple data sources, data collection, and analysis procedures, research methods, and inferences that occur at the end of the study [212]. This study triangulates survey results with two distinct sets of interview data to provide informative summaries about the characteristics that are perceived to contribute to the effectiveness of pedagogical tools and to the widespread adoption of these tools.

The analysis consists of classifying, categorizing and summarizing the results. The cross-category summary in Chapter 7 presented results of male and female respondents, college and secondary school teachers, those using the tool and those choosing not to use the tool. The aggregation summary compared responses of those evaluating tools in specific tool categories: Microworlds, Libraries, IDEs, and Visualization Tools. This analysis is designed to investigate the commonalities that exist in both the cross-category and aggregation survey responses and are supported by interview data.

8.2 Analysis Presentation

Several steps were involved in the preparation of the analysis summary.

1. The survey data were reported in Chapter 7 question by question for both the cross-categories and the aggregation of tools.

2. The interview data were reviewed for common phrases, patterns, and perceptions.

3. A list of common characteristics and patterns was formed from the survey data and interview results. From that list, several major themes developed. Each characteristic in the list was classified under one of the following major themes:

    - Reasons for choosing the tool: There are common reasons teachers choose to initially use the tool.
    - Manageable Environment: The tool was not difficult to use or install, simplified the mechanics of programming, and was neither too restrictive or too complicated for an introductory programming course.
    - Active Learning: The tool supports an interactive environment where students are actively engaged in the learning process.
    - Good First Experience: Students enjoy using the tool and programming is introduced in an enjoyable way through the use of the tool.
    - Visual Environment: The tool supports some form of graphical components or visualization techniques.
    - Flexible Environment: The tool engages many levels of learning and can be used throughout the course.
    - Subsequent Courses: The tool is a solid introduction to subsequent computer science courses and allows students to transition to these courses seamlessly.
    - Programming Activities: The tool provides support for programming activities.
    - Tool Resources: The developer, tool, or community, provides resources for using the tool in an introductory programming course.
    - Teaching: The tool eases and promotes the teaching of programming.
    - Learning: The tool eases and promotes the learning of programming.

4. A summary table was created organizing tool characteristics within these themes.

5. The survey summary data reported in Chapter 7 were reviewed for characteristics that the majority of respondents perceived to contribute *significantly* to the effectiveness of the tool. Only the majority percentages are recorded in the summary table. If the cell does not include a percentage, the percentage was less than 50%.

    - The cells identifying respondents that perceived the corresponding characteristic to contribute to the effectiveness of the tool were indicated in the summary table by color-coding the cells pink.
    - The cells identifying respondents that perceived the corresponding characteristic to be inadequately supported by the tool were indicated in the summary table by color-coding the cells green.
    - The cells identifying respondents that perceived the corresponding characteristic to contribute *significantly* to the effectiveness of the tool but are still inadequately supported by the tool were indicated in the summary table by color-coding the cells gray.

The complete color-coded summary table for this analysis appears in Appendix E.

8.3 Addressing the Research Questions

In summarizing the responses as they reflect the major themes listed in Section 8.2, the following research questions are addressed:

1. What influences the use of pedagogical tools in the introductory programming class? Specifically, what is the primary reason a teacher chooses to use a particular tool in an introductory programming class?

2. What are the perceived characteristics of an effective pedagogical tool used in an introductory programming course (effective is defined as: eases and promotes the teaching and/or learning of programming)?

3. What are the perceived characteristics of a pedagogical tool that hinder (or get in the way of) teaching and/or learning in an introductory programming course?

In addressing these questions, the survey data is summarized both within questions and across questions integrating interview results were appropriate. The results of the study may be referenced as specific tool categories (microworlds, libraries, IDEs, and visualization tools), as total respondents, or according to gender or teaching level (secondary school verses college). Throughout this analysis, the researcher takes the liberty of offering opinions on the relevance of the findings.

8.3.1 What Influences the Use of Pedagogical Tools in the Introductory Programming Class?

Using a pedagogical tool in an introductory programming class involves choosing the right tool to use for the appropriate pedagogical goals. The computer science teacher makes a conscious choice to use a specific pedagogical tool in an introductory programming course. The choice is influenced by the perceived effects the tool has on learning and on teaching.

8.3.1.1 Tool Choice

This study confirms several reasons for tool choice. Many of these reasons are based on the teachers' perceptions of the students' learning experiences in the programming course. The characteristics contributing to the initial adoption of a tool in an introductory programming course include:

- introducing programming in an enjoyable way
- improving first-time programmers experience
- positively affecting subsequent programming experiences
- easy to learn

Although these reasons do not indicate that specific pedagogical goals are being addressed, the researcher acknowledges that they are characteristics of the tool that encourage retention in the course and in the discipline.

The survey data indicate, in all cases, that the primary reason a teacher chooses to use a tool in an introductory programming class is *to introduce programming in an enjoyable way*. It is certainly not surprising that students should "enjoy" the experience of learning to program. And, certainly the goals of the tool developers are intended to support this [135, 202]. The interviews with tool developers confirm that the design of many of the pedagogical tools in this study were influenced by a need to modernize the introductory programming course to make it more appropriate and enjoyable for today's student. The developers also agreed that the tool was designed to provide a framework in which deep programming concepts can be taught without the overhead of language details.

Regardless of gender or teaching level, the majority of the respondents in this study perceive that the pedagogical tools used in their introductory class *improves the first-time programmers experience* allowing the student to have an initial encounter that *positively affects subsequent programming experiences*. This was confirmed by respondents using microworlds, libraries and visualization tools in their introductory programming classes. *Introducing programming in an enjoyable way* was perceived to be a characteristic contributing *significantly* to the effectiveness of the tool being evaluated by the respondents, regardless of gender or teaching level.

8.3.1.2 Recommendations from Other Computer Science Educators

Knowing that one wishes to introduce programming in an enjoyable way does not identify exactly why a teacher chooses to use one particular pedagogical tool over another. While tool choice is influenced by the reasons stated above, this study also confirms that the primary influence for initially choosing to use a pedagogical tool in an introductory programming course is the recommendation of other computer science educators. This recommendation may be made without specifically stating the pedagogical goals that are addressed by using the tool. The respondents directly involved in this study and those computer science educators involved in the various on-line communities that are addressed in this study are experienced educators. The researcher is confident that when one of these computer science educators is influenced by a recommendation, or offers a

recommendation, the recommendation is based on sound pedagogical considerations.

The results of the survey indicate that the primary influence in deciding to use a particular pedagogical tool is the *recommendation of other computer science educators*. Those evaluating microworlds, libraries, and IDEs confirmed this, as did the responses of the secondary school teachers participating in the study. The teacher interviews confirmed that the perceived credibility of those recommending the tool was important and that the primary sources of the "credible" recommendations were computer science education on-line *communities*. The *communities* were identified as the Special Interest Group on Computer Science Education listserve (SIGCSE Announce), the SIGCSE Annual Symposium, and the Advanced Placement Computer Science Electronic Discussion Group (AP CS EDG).

Most of the tool developers that were interviewed are linked into both mailing lists, SIGCSE Announce and the AP CS EDG. Although a many of the tool developers may not necessarily be active participants in the electronic discussions, either the tool developer or a member of the tool development team is aware of any communication that develops on the lists about their pedagogical tool. SIGCSE Announce reaches more than 2600 members, mostly college educators. The AP CS EDG reaches approximately 1950 members, both high school and college computer science educators. The high school members of the AP CS EDG primarily teach AP Computer Science, a course that is equivalent to a first semester college level computer science course. The students enrolled in AP Computer Science are expected to take the AP Computer Science Examination. Passing this exam will earn the student credit and/or placement in many colleges.

All introductory programming teachers do not teach AP Computer Science and may not be members of the AP CS EDG. It is this non-AP K-12 population of computer science educators that may not be linked into hearing about the newly developed pedagogical tools that can enhance and improve teaching the teaching of and learning of programming. There are also college educators that are not members of SIGCSE

Announce and do not attend the annual SIGCSE Symposium.

The researcher addressed these issues in the interview portions of this study. The teacher interviewees were asked "If you really believe in the effectiveness of a tool and you wanted to tell the world of introductory programming teachers about the tool, what mechanisms would you use to communicate with these teachers, knowing that all introductory programming teachers ARE NOT hooked into listserves?" As an alternative means of sharing information about pedagogical tools, the college educators that were interviewed suggested the Consortium for Computing Science in Colleges (CCSC) conferences. There are ten conferences annually, one in each of the ten regions of the United States. Most conferences welcome and encourage secondary school teachers to attend. Both the secondary school teachers and the college teachers that were interviewed suggested the Computer Science Teachers Association (CSTA) as an alternative means of informing K-12 computing teachers about pedagogical tools. The Computer Science & Information Technology (CS & IT) Symposium was the most mentioned CSTA resource. Although conducting workshops at the SIGCSE Symposium was a resource for many of the tool developers, few of them have personally outreached to the secondary school community at the CS & IT Symposium.

The interviews with the tool developers confirmed that teacher recommendation (word-of-mouth) was the primary source of advertisement. Teachers and tool developers agreed that high school teachers are usually departments of size one, "not well-organized above the local level," and therefore more challenging to reach than are college computer science educators.

Although recommendation was the primary influence for secondary school teachers to choose a particular tool, the majority of college respondents indicated that a task-technology fit that compliments their present teaching style was an impetus for deciding which pedagogical tools to incorporate into their courses. Regardless of teaching level, those interviewed confirmed that they choose pedagogical tools that will allow them

to teach what they are teaching more effectively and more efficiently. "The tool has to have some kind of added value beyond just that its nifty to look at or cool to use."

The review of literature completed for this dissertation confirms that all of the tool developers or members of the development teams provide information about the pedagogical tool through articles in professional journals yet only 9% of the total respondents to the survey read about the tool in a professional journal.

8.3.1.3 Teaching Experience

The comfort level of the teacher with the subject material and with teaching the subject material may have a great deal to do with using a pedagogical tool effectively and efficiently. More than 84% of the total survey respondents have taught the introductory programming course five or more times and more than 43% of the respondents have used the tool that they were evaluating five or more times. Comfort level with the course content and with teaching the course content contributes to the reaction that a major influence on which tool to use, as described by a teacher interviewee, is dependent on the tool replacing something that the tool does better (improving functionality) or adding to what is already being done (adding functionality). The teachers interviewed speak from experience when they confirm that a good tool will help with better, different, and/or deeper learning outcomes.

8.3.1.4 Learning the Tool

Before introducing the tool to the students, the teacher may wish to learn about the tool. There are two aspects of this learning process: the tool's "nuts and bolts" (how the tool works), and how to best incorporate the tool to benefit from the pedagogical goals for which the tool is designed. The easier of these two aspects is learning how the tool works. The survey indicates that more than 50% of those responding had no formal training in the use of the tool. They knew nothing about the tool and learned by experimentation. This was echoed by respondents of microworlds, IDEs, and visualization tools. About 70% of the total repsondents agreed that this training was enough. The teachers interviewed

confirmed that, in the field of computer science, teachers learn tools by "fiddling around" with them. This was qualified by noting that teacher experience helps with that learning process and that "fiddling" may allow the teacher to learn *how* the tool works but won't offer much in using the tool effectively within a pedagogical framework. The *why* and *when* are not necessarily the results of the *how*.

As discussed in Section 4.2, people learn in different ways. Learning depends on experience but also requires reflection, developing abstractions, and active testing of these abstractions [233]. This applies to teachers as well as students. The learning cycle usually starts with the "Why", to motivate the material and to give the "big" picture. Although many computer science teachers may learn the "nuts and bolts" of the tool by reading manuals (19%), using tutorials provided by the tool (18%), or "fiddling around," face-to-face workshops allow for active engagement of the participants where the tool and its pedagogical strengths (the "Why") can be observed. Although many of those interviewed learned tools without formal training, the majority felt that face-to-face workshops offered the greatest return for the time spent learning the tool. Teacher workshops may be impractical for some, schedule-wise or financially. An alternative mentioned by the interviewees was short video clips on the tool use and its pedagogical strengths. Web-based training would be agreeable if it didn't require a great deal of time.

Some of the teachers interviewed were comfortable learning the tool *with* their students though they also confirm that they are learning more about the *how* and not much about the *why* when they do this. The second time teaching with the tool may focus more on the *why*. Again, the comment of the interviewee stresses the goal: The tool has to have some kind of added value beyond just: "it's nifty to look at."

Often times when teachers are asked why they do not use a certain tool in their class, the response is that they do not have time to learn the tool and/or they can not afford to the time to use the tool in their class because it would take away from the curriculum that they must cover. Time seems to be a focus in a teacher's reaction to incorporating a new

or different pedagogy into their programming course. Time is valuable and there is little of it to spare. This is of primary importance in the secondary schools where the computer science teacher has multiple preparations and little time to prepare. As one teacher interviewee states, "if learning the tool takes too much time, then I am not going to use it." Although all of those teachers interviewed are familiar to the reactions citing time issues, the interviewees all agree that good tools will end up making time. As was stated in the teacher interviews:

- It doesn't matter if lots of material is covered if the students don't understand and absorb it.
- If a tool helps with better learning outcomes, it ends up saving time, giving the teacher time to spend on different or deeper concepts thus resulting in better learning outcomes.

Choosing the right tool to use for the appropriate pedagogical goals involves learning the tool and learning about the tool. The interviewees' recommendation for successful tool integration is for the teacher to *learn the tool* and *learn about the tool* before introducing it in the educational environment and integrating it into the curriculum.

8.3.2 What Characteristics are Perceived by Teachers to Contribute to the Effectiveness of the Tool(s) they Choose to Use in an Introductory Programming Course?

Several major themes categorize characteristics perceived by teachers to contribute to the effectiveness of a pedagogical tool.

- Manageable environment
- Active learning
- Visual environment
- Flexible Environment
- Adequate preparation for subsequent courses

- Programming activities
- Tool resources
- Eases teaching
- Eases and promotes learning

The researcher acknowledges that each of these themes may have a different level of importance for each individual teacher. The researcher also acknowledges that the teacher is the connecting link between the tool and the students [165, 135] and that it is not only the tool, but how, when, and why the teacher incorporates tool into the curriculum that contributes to its effectiveness in the classroom setting. This research was designed to gather information about tool characteristics that contribute to the effectiveness of pedagogical tools used in introductory programming courses as perceived by the teachers using the tools in these courses. The results of this study, when communicated to other computer science educators, provide information that may serve as a guide in the adoption of pedagogical tools in an introductory programming course. The results also provide tool developers with information that may contribute to the development of a more effective tool that will be adopted on a larger scale. The major themes that categorize the characteristics that are perceived to contribute to the effectiveness of a pedagogical tool will be discussed individually.

8.3.2.1 Manageable Environment

The majority of all survey respondents, regardless of gender and teaching level, indicate that the tool *supports an intuitive interface* and *has a learning curve that is not too steep* are characteristics that contribute to the effectiveness of the tool. A majority of those teachers evaluating microworlds and visualization tools confirmed this. Microworlds, IDEs and visualization tools were perceived to *simplify the mechanics of programming* by a majority of those evaluating these tools. *Simplifying the mechanics of programming* was perceived to contribute *significantly* to the effectiveness of the tool by most survey respondents. *Simplify* is the key. Interviews with tool developers' confirmed a

development goal of "allowing students to understand deeper concepts *easily and quickly*." Although a goal of the tool developers and a characteristic perceived by teachers to contribute to the effectiveness of the tool, *simple and manageable* is not always a characteristic that is evident. One teacher interview revealed that the students were excited about the tool and they wanted to use it at home but it was a nightmare to setup.

When the interviewed teachers were asked "What information or suggestions would you like to share with the tool developers?", the consensus was that the tool should be "plug and play", extremely easy to get up and running. "The ideal would be something that's all self-contained and works for whatever the purpose is."

Those respondents evaluating microworlds, libraries, and IDEs believe that *technical difficulties with installation* was a tool characteristic that interfered with learning.

When asked what characteristics should be better supported by the tool developers, microworlds and IDEs were perceived by the survey respondents to be too restrictive or too big (respectively) for introductory programming classes. The researcher acknowledges that some tools are specifically designed to be "small". Quoting a tool developer, "We tried to keep it very simple so that it would run on minimal equipment. That's probably the biggest value. Its purpose is quite limited by design. We had a few things we wanted to focus on. So it does, what it was designed to do."

Some tools are used in introductory classes but are intended for software professionals: Professional IDEs are designed to help software developers write programs more quickly and produce better quality code. This is not the focus in an introductory programming course. Some observations indicate that the professional IDE is challenging to students but the challenge actually inspires he students' "can-do" attitudes. They also observed that after using a professional IDE, most of their students showed a sense of confidence [41].

The teachers involved in this study believe that a manageable environment contributes to the effectiveness of the tool. Having a manageable environment allows a teacher to

concentrate on higher level programming skills without focusing on technical issues and installation difficulties. A manageable environment simplifies the mechanics of programming allowing the teacher to focus on higher level problem solving skills. The survey data, supported by the teacher interviews, propose that a manageable environment is ideally an all-in-one tool that is "plug and play".

8.3.2.2 Active Learning

Active learners learn by trying things out and working with others. *Uses active learning techniques* is one of the seven principles based on research on good teaching and learning offered by Chickering and Gamson [42] and one that is supported by the respondent in this study. A majority of the survey respondents, regardless of gender or teaching level, considers active learning as a characteristic that contributes to the effectiveness of the tool evaluated. The characteristics related to active learning in this study include:

- supports an interactive environment
- supports students' active engagement in learning activities

Supporting students' active engagement in learning activities was perceived to *significantly* contribute to the tool's effectiveness by all respondents regardless of gender or teaching level. Past research confirms that students learn best when actively engaged in the learning process [42, 202].

Although those evaluating microworlds perceived a characteristic that contributed to the effectiveness of the microworld to be that the microworlds *encouraged students to learn through discovery*, this was not a characteristic perceived by the respondents to be common to the other tool categories.

The teachers involved in this study believe that students should be actively engaged in the learning process and that a pedagogical tool used in an introductory programming course should support a student's active involvement. Although some tools may be used for demonstration purposes only, teachers perceive student involvement to be essential in the

learning process. Although supporting active engagement in learning activities is a characteristic that teachers perceive to contribute to the effectiveness of a tool, the researcher acknowledges that it is the teacher's methodology that initiates active learning in the class environment. An effective tool encourages this methodology.

8.3.2.3 Visual Environment

The Felder-Silverman Learning Style Model classifies students according to four dimensions [214]. One of these dimensions is visual learners (prefer pictures, diagrams, flow–charts) $\longleftrightarrow$ verbal learners (prefer written or spoken explanations). A previous study done by Allert found that reflective and verbal learners experienced more success in CS1 and CS2 courses than did those students classified as active or visual learners [8]. Most recently, the developers of pedagogical tools have dealt with this in a variety of ways. In all four tool categories, a majority of survey respondents have perceived some characteristic dealing with graphics or program visualizations as a characteristic that contributes to the effectiveness of the tool. The characteristics supported vary. Libraries provide graphics or media rich environments; IDEs support visualization of OO concepts; Visualization tools provide multiple views at once; Microworlds support visualization of program state and state changes. The total respondents considered *supporting the visualization of OO concepts* to contribute significantly to the effectiveness of the tool; they also believed that this concept was not adequately supported by the tool being evaluated.

It is interesting to note that a majority of college respondents did not focus on graphics or visualizations as contributing to the effectiveness of the pedagogical tool being evaluated but they did perceive that *supporting the visualization of OO concepts* was not adequately supported by the tools evaluated.

8.3.2.4 Flexible Environment

The Logo programming language is often described as having a low floor and high ceiling: it is easy for novices to get started (low floor) and possible for experts to work on

increasingly sophisticated projects (high ceiling). Resnick and Silverman have added a third dimension to their work in designing construction kits for kids: wide walls. They design technologies that support and suggest a wide range of different explorations [191]. "Low floor, high ceiling, wide walls" was quoted in both the teacher interviews and interviews with the tool developers and is supported by the survey results as a characteristic that significantly contributes to the effectiveness of a pedagogical tool.

A majority of respondents, regardless of teaching level or gender and regardless of tool category perceive that *allowing for use at many levels of learning* is a characteristic that contributes to the effectiveness of the tool. Respondents perceive microworlds to *engage* students of different ability levels, attract a diverse group of students, and allow for students to work at the level of their (the student's) choice. These same respondents believe that microworlds do not adequately allow for use at many levels of learning.

A tool that *can be used for the duration of the course* is a characteristic that is perceived to contribute *significantly* to the effectiveness of the tool by the majority of respondents regardless of gender or teaching level in three of the four tool categories: libraries, IDEs, and visualization tools. This is a characteristic that was not perceived as contributing to the effectiveness of microworlds.

The teachers involved in this study perceive a flexible environment as a characteristic that contributes to the effectiveness of a pedagogical tool. Ideally, a flexible environment introduces students to programming in a gentle and appealing way but also allows them to progress to more advanced features within the environment at appropriate points in the course. A flexible environment allows for a seamless transition to subsequent courses. It has "low floors, high ceilings, and wide walls" [191].

8.3.2.5 Subsequent Courses

When asked for the name of the programming course in which the pedagogical tool is being incorporated, the responses included such titles as:

- Computer Science 1
- Introduction to Programming
- Introduction to Object-Oriented Programming
- Introduction to Java
- Fundamentals of Computer Science
- Advanced Placement Computer Science

The question that was not asked in the survey was, "After completing this course, do you expect the students to continue their formal education by taking subsequent courses in computer science?"

One might assume that a course that has a '1' at the end of its name would lead to a second course: Computer Science 1 followed by Computer Science 2. One might also assume that an Introductory course is followed by an Intermediate course in the same discipline and that an Advanced Placement Computer Science course may earn a student credit in the college computer science department thus placing the student in a higher level computer science course in the college setting. Another assumption may be that since 23.5% of the total respondents designated their courses as *requirement for the computer science major*, at least 23.5% of the courses are preparing the students for subsequent computer science courses. These assumptions may or may not be correct.

To get a better picture of what might be expected from students taking these computer science courses, the responses of secondary school teachers and of college teachers are looked at separately.

At the secondary level, 71.2% of those responding indicate the course is an elective course. In most states in the United States, computer science is not a required disciple and so any course in computer science would be listed as an *elective* and that designation gives little information. However, 47.2% (59/127) of the secondary respondents indicate that the

course is designated as Advanced Placement. By the nature of the Advanced Placement Program, the students successfully completing the course are expected to be prepared to continue on to subsequent computer science courses at the college level.

At the college level, 70.2% (40/57) of the respondents indicate that the course is designated as a requirement for the computer science major. This would imply that the course is designed to prepare the student for the next sequential course in the computer science major.

Considering both categories (majors requirement on the college level and Advanced Placement on the secondary level) at least 53.8% of those teachers responding are teaching courses that will prepare the students for subsequent courses in computer science.

The majority of all respondents regardless of gender or teaching level perceived: *After using this tool students are ready to extend their knowledge by continuing in computer science curricula* to be a characteristic that contributed to the effectiveness of the tool. This is supported by the course designation as described above. The majority of those evaluating microworlds and IDEs also perceived this to be a characteristic contributing to the effectiveness of the tool.

Those evaluating libraries perceived that the use of libraries in the introductory programming course helped with the retention *in* the course and that this *significantly* contributed to the effectiveness of the tool.

There was a consensus among those responding about the students' transition to subsequent programming courses. All respondents regardless of gender or teaching level perceived that *the transition to a more robust programming environment* was not adequately supported by the tool. Those evaluating microworlds believed that the microworlds did not adequately support *transition to the real world* and did not adequately support *good object-oriented style* believing that good object-oriented style may be distorted by pragmatics and limitations of the tool.

The teachers involved in this study all use pedagogical tools in an introductory programming course. Inevitably, there is some course that follows an introductory course. Ideally, a pedagogical tool will support a seamless transition to subsequent courses. Students will not be required to *unlearn* concepts due to the inadequate or misleading representation of these concepts within the tool's environment.

8.3.2.6 Programming

This study focuses on the pedagogical tools that teachers use in their introductory programming classes. How the tool supports programming concepts is important in deciding the characteristics that contribute to the effectiveness of the tool. A sub theme in this category surfaced as *programming small pieces*. A majority of those evaluating IDEs and visualization tools perceived that the *step-by-step evaluation of single programming* statements was a characteristic that contributed to the effectiveness of the tool. College respondents perceived supporting *incremental development* contributed significantly to the effectiveness of a tool. The total respondents, regardless of gender, perceived that *supporting the testing of the individual components* contributed significantly to the effectiveness of the tool. JUnit and JamTester are both tools that support unit testing and were both tools that were listed in the "other" choice of tools being used in the introductory programming course as documented in Table 3, Section 7.2.

A majority of respondents, regardless of teaching level, perceived *supporting debugging in an easy way* to be a characteristic that contributed to the effectiveness of the tool. Those evaluating IDEs and visualization tools felt that this contributed *significantly* to the tool's effectiveness.

A majority of respondents, regardless of gender or teaching level, perceived *supports comments and documentation* as a characteristic that contributed to the effectiveness of the tool but believes that the tool does not adequate provide meaningful error messages. Secondary school teachers and those evaluating IDEs believe that the tool did not adequately prevent syntax errors.

As mentioned earlier, supporting an interactive environment is perceived to be a characteristic that contributes to the effectiveness of the pedagogical tool by the majority of all respondents, regardless of gender, teaching level, or tool category being evaluated. Event-driven programming is a paradigm in which *events* trigger *actions*. Mouse clicks may trigger certain responses in the program outcome while key-presses may trigger other responses. Event-driven programming supports an interactive environment. A majority of those evaluating microworlds perceived *supports event-driven programming* to be a characteristic that contributed to the effectiveness of the tool. Those evaluating libraries perceived this characteristic to contribute *significantly* to the effectiveness of the tool. College respondents, female respondents, and those evaluating visualization tools believed that this characteristic was not adequately supported by the tool being evaluated.

Programming involves several activities, e.g., learning the language features, program design and implementation, testing, and program comprehension [5]. Pedagogical tools designed to be used in an introductory programming course should adequately support these activities. The teachers involved in this study believe that the tool should support the ability to test individual components, support debugging in an easy way, and give immediate feedback about errors. The survey results indicate that *providing meaningful error messages* is not adequately supported by the tool and, in most cases, event-driven programming is not adequately supported by the tool.

8.3.2.7 Tool Resources

A tool is of little value if not used. As discussed previously, a teacher learns *how* to use a pedagogical tool ("the nuts and bolts") in a variety of ways. Incorporating the use of the tool in the introductory programming class does not necessarily result from learning how the tool works. The majority of the survey respondents, regardless of gender, teaching level, or tool category being evaluated, perceived *the tool can be used regardless of the textbook chosen* as a characteristic that contributed *significantly* to the effectiveness of the tool. However, secondary school teachers, college teachers, those evaluating microworlds, and those evaluating IDEs believed that *textbooks that use the tool* is a characteristic that

is not adequately supported. The majority of all respondents perceived that characteristics contributing to the effectiveness of the tool included the availability of tutorials and teacher's guides and the ability to find materials through on-line searches.

All respondents, regardless of gender and teaching level, believe that the tool does not adequately support the following tool characteristics:

- A mechanism for sharing materials is provided by the author or the community.
- Instructor resources (PP presentations, sample problems and labs, syllabus) are provided by authors or community.

The teacher interviews provide more depth to this response:

- "If you don't know how you are going to use the tool, it will be more of a disaster than anything else."
- "They(teachers) need to have a pedagogical framework for what the tool is suppose to be able to do."
- "Teachers need pedagogical examples of why the tool is strong."

Many of the tool developers offer some form of teacher resources that are available on-line [6, 28, 62, 77, 82, 83, 104, 107, 108, 152, 166] and most tool developers offer some means of communication through the tool's web site or the author's web site. *Technical support provided by the authors or the community* is a characteristic perceived to contribute to the effectiveness of the tool by a majority of the respondents regardless of gender or teaching level. This characteristic was considered to contribute *significantly* to the tool's effectiveness by those evaluating IDEs.

Although many teachers learn *the how* of the tool by "fiddling around", auxiliary resources can provide the teacher with a pedagogical framework in which to incorporate the tool. As confirmed by teacher interviews, time is a valuable resource. Developing meaningful activities, challenging projects, and engaging presentations is time-consuming.

Providing resources for tool use provides guidance for *the why* and *the when* that support sound pedagogical goals.

8.3.2.8 Teaching

Programming is a key objective in most introductory computing classes and it is a skill that is both challenging for teachers to teach [91, 94, 124, 140, 200, 231] and difficult for students to learn [18, 91, 94, 116, 149, 151, 177, 214, 225]. For students to achieve better comprehension of programming and to enhance understanding of programming concepts, teachers of introductory programming have adopted a myriad of pedagogical tools. Some teachers incorporate the use of a single tool in the course while others use a variety of tools in a single course. Regardless of the tool category or number of tools used, the teaching of programming has been enhanced and improved by the inclusion of pedagogical tools. The teachers involved in this study use pedagogical tools in their introductory programming courses and will continue to use those tools in future courses. Table 42 confirms this. The Likert scale responses of those using the tool indicate the following:

- 96% of those teachers using a tool agree or strongly agree that they will continue to use that toold in future courses.
- 95% of those teachers using a tool agree or strongly agree that they will recommend the tool to other computer science educators.
- 88% of those teachers using a tool agree or strongly agree that using this tool in their introductory programming class eases and promotes the teaching of programming.
- 74% of those teachers using a tool agree or strongly agree that using this tool in their introductory programming class eases and promotes the teaching of object-oriented concepts.

The majority of all survey respondents, regardless of gender, teaching level, or tool category being evaluated, perceived that being *a good tool for classroom demonstrations* is a characteristic that contributes to the effectiveness of the tool. Finally, all interviewed

teachers agreed that the way they teach would change significantly if the tools were removed from their teaching repertoire.

The teachers involved in this study successfully integrate the use of pedagogical tools in introductory programming courses. The teacher interviews support a modification of Michael Kölling's hypothesis presented in Section 2.2: The difficulties and problems in teaching programming "could be overcome or reduced through the use of appropriate tools." The teachers involved in the study agree that an effective tool facilitates the teaching of the most critical topics in a programming course.

8.3.2.9 Learning

Teachers are the the most influential factor and the connecting link between the pedagogical software and the students [165, 135]. Engaging students is critical to deep learning [91] and today's learners are motivated in ways unlike previous generations. A majority of the survey respondents, regardless of gender, teaching level, or tool category being evaluated, believe that the tool *eases and promotes the learning of programming.* These same majorities perceived *helps with the understanding of program execution* to be a characteristic that contributes to the effectiveness of the tool. The Likert scale responses of those using the tool indicate the following:

- 87% of those teachers using a tool agree or strongly agree that using this tool in their introductory programming class eases and promotes the learning of programming.
- 73% of those teachers using a tool agree or strongly agree that using this tool in their introductory programming class eases and promotes the learning of object-oriented concepts.

A majority of those evaluating microworlds, libraries, and IDEs perceived *supports the understanding of abstract and complex concepts* to be a characteristic that contributes to the tool's effectiveness. Those using and evaluating microworlds indicated that the tool did not adequately *support abstraction* This was also a characteristic perceived by

secondary school teachers not to be adequately supported by the tool. Although a majority of secondary school teachers perceived *the tool embodies ideas that support core programming concepts* to be a characteristic that contributed to the tools effectiveness, they also felt that this characteristic was not adequately supported by the tool. A majority of those evaluating microworlds perceived *supports a concepts first approach* to *significantly* contribute to the tool's effectiveness.

A tool that supports the *understanding of program execution* and the *understanding of abstract and complex concepts* eases the learning of programming. These characteristics are supported in a variety of ways by the tools belonging to the different tool categories.

8.3.3 What are the Perceived Characteristics of a Pedagogical Tool that Hinder (or get in the way of) Teaching and/or Learning in an Introductory Programming Course?

The survey queried teachers about characteristics of the tool that are perceived to hinder the learning or the teaching of programming. The majority of respondents, regardless of gender, teaching level, or tool category being evaluated, indicated that the tool evaluated had *no significant negative aspects* related to the tool environment, errors, or support. Small majorities of those using the tool indicated that the common characteristic across tool categories perceived to hinder teaching or learning is the generation of *cryptic error messages* when using the tool. The interviews with the teachers also pointed to technical difficulties as interfering with the learning process:

- "The kids couldn't save their work..."
- "It's hard to ramp up on."
- "I couldn't get it to consistently work."
- "The program just crashes."

The survey results indicate that 97% of the teachers using a pedagogical tool in their introductory programming class will continue to use that tool in future courses. Neither teaching or learning is hindered enough to discourage the use of the tool.

82% of the non-users answered the question focusing on the perceived negative aspects of the tool being evaluated. Looking only at the non-users responses, the following characteristics, with percentage of non-users choosing the characteristic, were considered to hinder teaching or learning:

- Students have technical difficulties (29%).
- The tool is too big for introductory classes (29%).
- The learning curve is too steep (25.8%).
- The transition to real-world programming is difficult (25.8 %).
- No easy way to debug (23.3%).
- Technical difficulties with installation (23.3 %).

Although negative aspects are indicated by teachers, users and non-users, when considering all survey respondents, more that 55% of those responding perceived the tool being evaluated to have no significant negatives.

Computer science teachers using pedagogical tools in introductory programming classes do so to make abstract concepts more concrete for the student. But it's not the tool, it's how and when the teacher uses the tool that supports student learning. The researcher agrees with an interviewed teacher's statement: "If you don't know how to use the tool and why you are going to use the tool, it will be more of a disaster than anything else."

Chapter 9

Summary and Future Work

> *The basic question is to decide what to do: either to develop a tool for existing teaching and learning practices, or change teaching and learning practices by developing a new tool* [129].

The goal of this research was to investigate the characteristics that are perceived to contribute to the effectiveness of pedagogical tools used in introductory programming classes. This study was motivated by the need to look at changes in pedagogy that can address the challenges that teachers and students face in an introductory programming course. In particular, this study investigates the motivations for the integration of pedagogical tools in introductory programming classes in order to share information with tool developers about teachers' perceptions of effective tools in the event that they plan any improvements, modifications, or innovations and to encourage an efficient and effective wide spread adoption of the tool information to all branches of the computer science education community.

My hope is that this research makes some small contribution to the tool developers repertoire when designing a pedagogical tool and to the programming teacher's motivations for choosing a tool to use in an introductory programming course.

The result of this study provides perceptions of a limited population of introductory programming teachers. The teachers surveyed are most likely at a higher level of professional involvement in professional conferences, professional development workshops and activities, and discussion groups focusing on computer science education. The teachers involved in this study are experienced practitioners in their field. They have successfully used pedagogical tools in introductory programming courses. Sharing their perceptions of tool characteristics can contribute to the adoption of a tool by other

programming teachers. As Rogers states, "The [individuals'] perceptions of the attributes of innovations, not the attributes as classified by experts or change agents, affect its rate of adoption [195]."

The analysis of the interviews shows evidence that those interviewed drew on their experiences when responding to the questions. The feedback provided by those interviewed contribute additional insight to and support for the survey results.

9.1 Addressing the Hypotheses

The following hypotheses were stated in Chapter 3 and are addressed in this chapter.

1. Using pedagogical tools in introductory programming classes eases and promotes the learning of programming.

2. Using pedagogical tools in introductory programming classes eases and promotes the teaching of programming.

3. Effective pedagogical tools used in introductory programming classes have common characteristics.

4. Tools that teachers consciously choose NOT to use in introductory programming classes have common characteristics.

5. Teachers initially choose to use a pedagogical tool because of a perceived task-technology "fit."

9.1.1 Using Pedagogical Tools in Introductory Programming Classes Eases and Promotes the Learning of Programming.

This hypothesis is supported by both the survey results and the interviews with the teachers. 87% of the teachers surveyed, who are using the tool being evaluated, agree or strongly agree with this hypothesis. 73% of the teachers surveyed, who are using the tool being evaluated, agree or strongly agree the the tool also eases and promotes the learning of object-oriented concepts. The interviews confirm that a good tool will help with better,

different, and/or deeper learning outcomes. The survey results indicate that *simplifying the mechanics of programming* was perceived to contribute *significantly* to the effectiveness of the tool.

Today's programming students are different from previous generations. They have grown up in a world of emerging technologies. They interact with information differently from previous generations. Programming courses integrating appropriate pedagogical tools help address the needs of this Nintendo generation by providing opportunities for active learning and by providing visual representations that facilitate the understanding of programming concepts and program execution. An effective pedagogical tool allows for a flexible programming environment that addresses the needs of students of various ability levels and learning styles. An effective tool provides learning experiences that encourage students to continue in the discipline with a seamless transitioning experience.

9.1.2 Using Pedagogical Tools in Introductory Programming Classes Eases and Promotes the Teaching of Programming.

This hypothesis is supported by both the survey results and the interviews with the teachers. 88% of the teachers surveyed, who are using the tool being evaluated, agree or strongly agree with this hypothesis. 74% of the teachers surveyed, who are using the tool being evaluated, agree or strongly agree that the tool also eases and promotes the teaching of object-oriented concepts. The interviews confirm that a good pedagogical tool will give the teachers (and students) something to allow them to deepen or further their knowledge more than they would have without the tool. They are then able to exxpand their knowledge with a constructivist kind of approach.

The interviewed teachers agreed that the way they teach would change significantly if the tools were removed from their teaching repertoire.

- I don't know of anything that would let me create and inspect things as easily as BlueJ.
- It would tremendously change. I probably would not even come close to using the

languages and things that I am using today.

- It would have to change. The tool really helped me and it really allowed the students to get a foundation for when we went to Java.
- A lot would change. I would have to introduce `public static void main(String[] args)` again. I think my students have a much better grasp of objects and classes because of BlueJ.

Today's programming teachers employ different pedagogies to accommodate different programming paradigms, different learning styles, and more diverse student populations. The use of pedagogical tools in introductory programming courses does not in itself make teachers teach better but teachers are the most influential factor and the connecting link between the tool and the student, the teaching and the learning. The recommendation of the programming teacher using the tool is also the most influential reason that tools are adopted by other computer science educators.

An effective pedagogical tool eases the teaching of programming by providing the teacher with resources that not only provide the *how* but explain the *why* and the *when* of the tool integration within the curriculum. An effective tool has sound pedagogical goals and provides an engaging environment devoid of technical difficulties.

9.1.3 Tools that Teachers View as Effective Pedagogical Tools Used in Introductory Programming Classes Have Common Characteristics.

In analyzing the data obtained from the survey and the interviews, several themes categorizing the characteristics that contribute to the tool's effectiveness (regardless of tool category) emerged. The themes and the related characteristics are listed below.

- Manageable Environment
  - Has an intuitive interface.
  - Has a learning curve that is not too steep.

- Active Learning
  - Supports and interactive environment.
  - Supports the students' active engagement on learning activities.
- Good First Experience
  - Introduces programming in an enjoyable way.
  - Students enjoy using this tool.
  - Improves firs-time experience.
- Visual Environment
  - Supports graphics.
  - Supports visualization of OO concepts and program state.
- Flexible Environment
  - Allows for use at many learning levels.
  - Can be used for the duration of the course.
- Subsequent Courses
  - Prepares the student for subsequent computer science courses and a more robust programming environment.
- Programming
  - Supports programming, testing, and debugging in small pieces.
  - Supports comments and documentation.
- Resources
  - Can be used regardless of textbook chosen.
  - Technical support is provided by the author or the community.

This is not intended to be an exhaustive list but is representative of those characteristics that are perceived to contribute to the effectiveness of a tool used in an introductory programming course by the teachers involved in this study.

The results indicate that the pedagogical tools perceived to be effective share common characteristics: they provide an environment that is manageable, flexible and visual; They provide for active engagement in learning activities and support programming in small pieces; they allow for an easy transition to subsequent courses and more robust environments; they provide technical support and resource materials. The results of this study also indicate that recommendations from other computer science educators have a strong impact on a teacher's initial tool choice for an introductory programming course.

9.1.4 Tools that Teachers Consciously Choose NOT to Use in Introductory Programming Classes have Common Characteristics.

The majority of all respondents, regardless of gender, teaching level, or tool category being evaluated found no *significant* negative aspects of the tool being evaluated.

The two characteristics that were mentioned across questions and tool categories and/or in the teacher interviews as characteristics that interfered with the learning process were:

- Cryptic error messages
- Technical difficulties
- Difficult transition to a more robust environment

As this study confirms, most of the pedagogical tools designed for introductory programming courses have no significant negative aspects. However, all tools are not appropriate for all educational environments. As one teacher interviewee puts it, "...Once you know what your goal is, then you can say this tool either does it or doesn't do it. Then it makes not only using the tool better but it makes selecting it or not selecting it easier."

9.1.5 Teachers initially choose to use a pedagogical tool because of a perceived task-technology "fit."

Although the teacher interviews support this hypothesis, this survey data do not. Only 37.4% of the total survey responses indicated that a *task-technology fit* was a reason for initially choosing to use the tool. This was, however, a majority choice for the college teachers (54.4%).

The results of both the teacher survey and the teacher interviews clearly underscore the importance of *teacher recommendation* when initially choosing a tool to use in an introductory class. Teachers choose to use a tool because of its perceived value *as determined by other computer science educators.* 61% of the total survey responses indicated that the tool was initially chosen to be used in an introductory programming course based on the recommendations from other computer science educators.

The respondents directly involved in this study and those computer science educators involved in the various on-line communities that are addressed in this study are experienced educators. The researcher is confident that when one of these computer science educators is influenced by a recommendation, or offers a recommendation, the recommendation is based on sound pedagogical considerations.

9.1.5.1 Information for Tool Developers and Teachers

Interviews with the teachers and the tool developers confirm that the primary resource for communicating with computer science educators is through the computer science educators' mailing lists. Approximately 4500 teachers are members of SIGCSE Announce or AP CS EDG.

In a one day's posting (January 11, 2009) on the AP EDG, the following responses to "I thought it (Alice) might be a great introduction with a broad appeal, and might help get the students get ready for the Java needed for AP...Does anyone else have any thoughts on the appropriateness of Alice for high school?" The responses include the following comments:

- "They (the students) spent hours at home working on it, just for fun"
- "Alice is most appropriate as a first language, be it in Middle School, High School, or University."
- "Students love beginning to play with it.
- "My juniors and seniors enjoy Alice very much. We use it in our one term intro course and it has been a very big hit."
- "..and my students loved it and learned a great deal about challenging ideas."
- "Alice is a little simplistic, but that helps keep it out of the uncanny valley."
- "It made for a good introduction to the idea of objects, but there was a lot of weirdness in it that made things hard. "
- "Alice is great for high schoolers"

What troubles the researcher is that the *why* is not being addressed in most of the above responses. There is no mention of pedagogical goals in either the question or the responses. Why is Alice appropriate for high school and for preparing the students for AP? Only three of the fourteen responses posted in this one day made any mention of the *why* or *why not*.

- "Our 9th graders will spend about 4-5 weeks on ALICE, then SCRATCH, move onto HTML, Python and JAVA. To me it's about exposure. This gives the students opportunities to determine were they want to branch off to i.e. AP CS, the graphics oriented curriculum with animation and the likes."
- "It is a great introductory language because you can teach all of the important programming concepts without getting bogged down with the syntax."
- "It is my opinion that the pedagogy is not adequately developed. (Some of my concern is based on the drag-and-drop environment which is both appealing to

> students and gets in the way of learning solid concepts)....Make sure that you are using tools that match your goals and intended pedagogy."

An earlier (similar) thread (August 2008) about the same tool (Alice) includes a thorough response from a member of the Alice development team. The response addresses the pedagogical goals of the tool as well as the intended audience.

> "... In the traditional introductory level course, we confront our students with at least four different pedagogical/logistical hurdles in the first couple weeks of the course. (1) We want them to start to develop an understanding of what computer science and programming is. (2) We want them to understand that syntax matters. (3) We want them to learn a new piece of software for being able to complete their assignments. (4) File management – how an operating system works ("I cannot find where I saved that file."). Alice helps to separate these hurdles and allows students to deal with only one (or possibly two) at a time.
>
> ...
>
> Recognize that Alice, as a tool, was designed specifically for the CS 0 (or Pre-CS 1) course. It was not designed to be a replacement for other excellent tools that support the novice programmers as they come to the CS 1 course. We believe that no one tool is a "one size fits all". Different teachers and different students in different courses likely need to use various tools. Also, experience with more than one tool is a good way to enrich a student's background and understanding..."

The researcher encourages computer science educators that respond to questions about the use of specific pedagogical tools to address the *why* and *when* in their responses. Many of those asking the questions are inexperienced teachers needing guidance. As this study confirms, most of the pedagogical tools designed for introductory programming courses have no significant negative aspects. However, all tools are not appropriate for all educational environments. A teacher asking for guidance in choosing a tool to use in an

introductory programming course might benefit more if given information about the pedagogical strengths and weaknesses of the tool. As one teacher interviewee puts it, "You have to keep your eye on the prize before you use anything. But once you know what your goal is then you can say this tool either does it or doesn't do it. Then it makes not only using the tool better but it makes selecting it or not selecting it easier."

The researcher also acknowledges that the members of the tool development teams are "lurking" out there and encourages them to offer their input to the threads that question the use of their tool(s) in introductory programming courses. The researcher encourages the developers to include the pedagogical goals of the tool in their responses, the *whys*.

As one of the teacher interviewees remarked, "The tool has to have some kind of added value beyond just: it's nifty to look at."

9.2 Related Work and Research

There is a considerable amount of research currently taking place on pedagogical tools used in the introductory programming course. Confirmation of this can be found in the ACM Digital library where more than thirty papers or workshops on Alice, more than ten on Greenfoot, and more than twenty on BlueJ that have been published or delivered in the last three years. There is no denying that educators are using pedagogical tools in their introductory courses and computer science education researchers are investigating different aspects of the tools. As mentioned in Chapter 4, most assessments are done by the developers of the environments and many assessments being conducted are more opportunity-directed than problem-directed. Although some tools clearly attempt to address different learning styles, there is little formal research that supports this, or even considers this as relevant.

The ACM Educational Council has recently developed a website, *Technology that Educators of Computing Hail*, to provide information about present technological tools used in and outside the classroom [78]. Although the site is in its alpha stage, it may

prove to be a valuable resource for information about pedagogical tools used in the introductory programming class.

Ni from Georgia Institute of Technology has investigated the factors influencing computer science teachers' adoption of curriculum innovations [165]. The results indicate that the decision to adopt an innovation was most significantly driven by teacher excitement, not the pedagogical value of the approach. Ni's study suggests that further work is needed on understanding what makes teachers become excited in a new approach. This study confirms that need. The information in this dissertation suggests that recommendations from other computer science educators are the primary catalysts for the adoption of a tool by teachers of introductory programming courses and not necessarily for the pedagogical value of the tool.

9.3 Contribution to Knowledge

This dissertation has discussed teacher perceptions of the characteristics of pedagogical tools that contribute to the tool's effectiveness when used in introductory programming classes and the motivations of teachers in choosing the tools to incorporate in their courses. The purpose of this study was not to produce conclusive results or to make inferences about the perceptions of the general teaching population but rather to create a list of characteristics that a tool developer may wish to address in the design or modification of a pedagogical tool used in the introductory programming course or a teacher may wish to address when considering the adoption of a particular pedagogical tool. Teacher perceptions provide a resource that can contribute to an improved design, delivery, and adoption of pedagogical tools for introductory programming courses.

The data collected in this study are based on teacher perceptions. This data reveal:

- Teachers are influenced by and value the recommendations of other computer science educators.
- When choosing a tool, emphasis on introducing programming in an enjoyable way is

more evident than the emphasis on the pedagogical value of the tool.

- An effective tool used in introductory programming classes provides an environment that is manageable, flexible, and visual; it provides for active engagement in learning activities and supports programming in small pieces; it allows for an easy transition to subsequent courses and more robust environments; it provide technical support and resource materials.
- The primary resources to reach teachers are SIGCSE Announce and AP CS EDG.
- Face-to-face workshops stressing the pedagogical strengths of the tool is a valuable resource for teachers.
- The ideal tool has "low floors, high ceilings, and wide walls [191]" and can be used for the duration of the course.

9.3.1 Information for the Tool Developers

Tutorials and manuals are valuable resources for learning how the tool works. A majority of the teachers learn the tool by "fiddling around" and using manuals and tutorials when necessary. What is important for the successful integration of the pedagogical tool is knowing the *why* and *when* of its use. Teacher interviews confirmed that there was great value in watching a demonstration of the tool or attending a workshop with the tool developer where the developer focused on the pedagogical strengths of the tool in addition to the "nuts and bolts" of how to use the tool.

SIGCSE symposium was the primary resource for the workshops. Not many secondary school teachers attend SIGCSE. The CSTA CS & IT Symposium is designed for secondary computing teachers. Tool developers might consider presenting workshops or demonstrations at this symposium or actively engage in other outreach to secondary school teachers.

Another effective method is to provide short video clips of the tool use and pedagogical strengths. Although some developers offer tutorials that are incorporated in the tool are

of great help in learning the *how*, to effectively use the tool, the *why* needs to be stressed.

The teacher interviews asked, "What do the authors of the tools need to know or to ask the teachers?" Most of those interviewed replied,"what works and what doesn't work". The researcher is aware that tool developers are receptive to feedback. Perhaps more can be learned by actively seeking feedback from the teachers.

9.3.2 Information for the Teachers

The interview results of this study clearly support the use of pedagogical tools in an introductory programming course. Those interviewed all agree that learning to use the pedagogical tool takes time. Time is valuable and there is little of it to spare. This is of primary importance in the secondary schools where the computer science teacher has multiple preparations and little time to prepare. Good tools will end up saving time. If a tool helps with better learning outcomes, it ends up saving time, giving the teacher time to spend on different or deeper concepts thus resulting in better learning outcomes.

It is not enough to know *how* to use the tool, the tool needs to add value to the course and fit the appropriate pedagogical pocket.

The tool developers are all receptive to comments and suggestions. Some tool developers host workshops and address the concerns and suggestions at these workshops. Teachers: If you have suggestions or concerns, let the tool developers know about them.

9.4 Future Work

The purpose of this study was not to produce conclusive results but rather to create an informal list of characteristics a tool developer may wish to address in the design or modification of a pedagogical tool. The study also provides information about the motivations of teachers in choosing tools to incorporate in their programming courses. The study did not analyze each individual tool nor did it focus on a particular category of tools. The purpose was to gather general information on the attributes shared by successful tools (as perceived by the teacher) regardless of the tool category.

Most research done in the area of pedagogical tools focuses on one specific tool and its effectiveness (proven or observed) or one category of tools (e.g. Visualization tools) and the effectiveness of this type of tool on student learning. This study investigates the effectiveness of tools across categories. It is not a quantitative analysis of tool effectiveness; it is a study of the perceptions of teachers using these tool in the introductory programming classes.

From this study, it is strikingly clear that teachers who are using pedagogical tools in introductory programming classes do so primarily because they perceive the tools to ease and promote the teaching and learning of programming. Typically they have become convinced of the tool's benefits from the recommendations of other computer science educators. Although it is not clear that these recommendations are being made based on the pedagogical strengths of the tool, the researcher is confident that when one of these computer science educators is influenced by a recommendation, or offers a recommendation, the recommendation is based on sound pedagogical considerations.

This study serves as a starting point for more formal research and as a resource for teachers, future took developers, and those involved with computer science curriculum development.

Research is needed to validate the teacher perceptions of the characteristics that contribute to the effectiveness of a pedagogical tool used in introductory programming classes. This study provides a listing of characteristics perceived to contribute to the effectiveness of a tool. The results of this study can be validated by a measurable assessment of the pedagogical tools based on these characteristics.

It would also be useful to investigate the teachers' knowledge about the tool when it is initially chosen to be used in the introductory programming classes. Are the learning objectives of the course a good match with the pedagogical strengths of the tool? To what extent does the tool meet the learning objectives of the course. These questions can be investigated using measurable techniques.

The results of Ni's study [165] suggest that "teacher excitement in a new approach drives adoption, while more organizational or social issues inhibit adoption." The results were based on workshop participants using specific pedagogical tools. Ni's study, together with the results of this research, confirm that the teachers' perceptions about a tool influence the adoption of the tool. Research that focuses on the adoption of tools based on pedagogical goals is needed. When the teachers choose to adopt a tool, do they understand the pedagogical strengths and weaknesses of the tool?

Echoing Kim Bruce, "It would help if a group of experts in educational research were to design experiments that will allow faculty to examine the success of the innovative approaches proposed for teaching Java in CS1 [32]."

Appendix A

Pedagogical Tools Survey for Teachers

Pedagogical Tools Survey for Teachers

1. General Information

I am conducting research on pedagogical tools used by computer science educators in introductory programming courses. These tools can be categorized in a variety of ways: Algorithm Visualizations, Algorithm Animations, Microworlds, Integrated Development Environments, Robots, Game Making Tools, etc.

I have tried many of them in my own classes, some successfully and some not. In an effort to identify the characteristics of an effective pedagogical tool, I am requesting that you complete a survey for the tool (or tools) that you use (or have used) in your programming course. I am interested in both your successful experiences and your unsuccessful experiences!

Some of the popular tools are listed below but there are hundreds of others.

Alice, BlueJ, Buggles and Bagels, CodeWiz, DrJava, DrScheme, Eclipse, GameMaker, Greenfoot, Gridworld, Greeps, JamTester, Java Task Force Graphics, JCreator, Jeliot 3, Jeroo, jGrasp, JHAVE, JPie, Java Power Tools, Junit, Karel J. Robot, LEGO Mindstorms, Media Computation, NetBeans, ObjectDraw, PigWorld, RAPTOR, Robotran, Scratch, XTANGO, ZEUS

If time permits, please complete one survey for each tool with which you have experience.

*** 1. Name of tool:**

Tool Used

If OTHER, please specify

2. Are you presently using this tool in your course?

- yes
- no

3. Why did you initially choose to use this tool in your programming course? Check all that apply.

- (A) to attract a diverse group of students
- (B) to increase my enrollment
- (C) to introduce programming in a more enjoyable way
- (D) recommended to me by other computer science educators
- (E) task-technology "fit"
- (F) tool complements my teaching style
- (G) trying to move away from a purely traditional(lecture) teaching approach
- (H) read about it in a professional journal
- (I) other (please specify) If listing more than one reason, please list each reason on a separate line.

Page 1

Pedagogical Tools Survey for Teachers

4. Of the reasons listed above, which ONE reason was the MOST influential for you?

	your choice
reason	

If your choice is "(I) other" and you listed more than one reason above, please clarify which is the MOST influential.

Pedagogical Tools Survey for Teachers

2. Tool Characteristics

Check the characteristics below that you believe contribute to the EFFECTIVENESS of this tool in the teaching and learning of programming in an introductory programming course.

A tool may have a characteristic that DOES NOT contribute to the effectiveness of the tool. You would NOT check that characteristic.

All characteristics are not applicable to all types of tools.

1. Check the tool characteristics that you believe contribute to the EFFECTIVENESS of this tool.

- ☐ (A) has an intuitive interface
- ☐ (B) has a learning curve that is not steep
- ☐ (C) supports an interactive environment
- ☐ (D) improves first-time programmer's experience
- ☐ (E) is flexible (allows use at many levels of learning)
- ☐ (F) embodies ideas that support core programming concepts
- ☐ (G) supports a concepts-first approach (not syntax)
- ☐ (H) supports abstraction
- ☐ (I) supports consistent metaphor (e.g. turtles, robots, visual representation of objects)
- ☐ (J) includes graphical components
- ☐ (K) does not restrict scenarios
- ☐ (L) restricts scenarios
- ☐ (M) incorporates storytelling
- ☐ (N) allows for easy transition to a more robust programming environment
- ☐ (O) other (please specify: if listing more than one characteristic, list each characteristic on a different line.)

Pedagogical Tools Survey for Teachers

2. Which of the characteristics checked above do you perceive to be most significant contributors to the EFFECTIVENESS of this tool.

Use the drop-down menus to choose the letter that corresponds to the characteristic in the list above. If possible, list your choices in order of importance with the most important characteristic listed first.

	Characteristics
1.	
2.	
3.	

(O) If you ranked O as 1, 2, or 3 and if you listed more than one characteristic for O, please use the space below to clarify the characteristic for the specified rank.

3. Which of the characteristics of those that you DID NOT check above would you like the tool to support (if any).

Use the drop-down menus to choose the letter that corresponds to the characteristic in the list above. If possible, list your choices in order of importance with the most important characteristic listed first.

	Characteristics
1.	
2.	
3.	

Other (please specify any characteristics NOT listed above that you would like this tool to support. Include your importance rank (1, 2, or 3))

Pedagogical Tools Survey for Teachers

3. Programming Characteristics

The characteristics listed below relate to programming.

Check the characteristics below that you believe contribute to the effectiveness of this tool in the teaching and learning of programming in an introductory programming course.

A tool may have a characteristic that DOES NOT contribute to the effectiveness of the tool. You would NOT check that characteristic.

All characteristics are not applicable to all types of tools.

1. Check the PROGRAMMING ENVIRONMENT CHARACTERISTICS that you believe contribute to the effectiveness of this tool.

- ☐ Question does not apply for this tool.
- ☐ (A) simplifies the mechanics of programming
- ☐ (B) provides multiple views at once (e.g. code window, animation, state)
- ☐ (C) supports incremental development
- ☐ (D) supports the visualization of the program state
- ☐ (E) supports the visualization of state changes
- ☐ (F) supports direct state manipulation
- ☐ (G) supports visualization of program code
- ☐ (H) supports visualization of OO concepts
- ☐ (I) supports visual representation of general programming concepts
- ☐ (J) provides an environment in which sophisticated problems can be solved
- ☐ (K) provides a media rich programming environment
- ☐ (L) Other (please specify: if listing more than one characteristic, list each characteristic on a different line.)

Pedagogical Tools Survey for Teachers

2. Check the TESTING, DEBUGGING, AND INTERACTION CHARCTERISTICS that you believe contribute to the effectiveness of this tool

- ☐ Question does not apply for this tool.
- ☐ (M) supports step-by-step evaluation of single programming statements
- ☐ (N) supports developing and testing of individual components
- ☐ (O) supports debugging
- ☐ (P) provides meaningful error messages
- ☐ (Q) gives immediate feedback about errors
- ☐ (R) prevents syntax errors
- ☐ (S) does not prevent syntax errors
- ☐ (T) supports comments and documentation
- ☐ (U) supports user-provided input data
- ☐ (V) supports an easy way of including event-driven programming
- ☐ (W) supports introduction of data structures
- ☐ (X) Other (please specify: if listing more than one characteristic, list each characteristic on a different line.)

3. Which of the characteristics checked above do you perceive to be most significant for the EFFECTIVENESS of this tool?

Use the drop-down menus to choose the letter that corresponds to the characteristic in the list above. If possible, list your choices in order of importance with the most important characteristic listed first.

	Characteristics
1.	
2.	
3.	

(L or X) If you ranked L or X as 1, 2, or 3 and if you listed more than one characteristic for L or X, please use the space below to clarify the characteristic for the specified rank.

Pedagogical Tools Survey for Teachers

4. Which of the characteristics of those that you DID NOT check above would you like the tool to support (if any).

Use the drop-down menus to choose the letter that corresponds to the characteristic in the list above. If possible, list your choices in order of importance with the most important characteristic listed first.

	Characteristics
1.	
2.	
3.	

Other (please specify any characteristics NOT listed above that you would like this tool to support. Include your importance rank (1, 2, or 3))

Pedagogical Tools Survey for Teachers

4. Learning and Teaching Characteristics

The characteristics listed below relate to learning and teaching.

Check the characteristics below that you believe contribute to the effectiveness of this tool in the teaching and learning of programming in an introductory programming course.

All characteristics are not applicable to all types of tools.

1. Learning

- [ ] (A) supports the understanding of abstract and complex concepts
- [ ] (B) supports students' active engagement in learning activities
- [ ] (C) engages students of different ability levels
- [ ] (D) allows students to work at levels of their choice
- [ ] (E) provides a natural platform for encouraging group work
- [ ] (F) encourages learning through discovery
- [ ] (G) helps understanding of programming execution
- [ ] (H) Other (please specify: if listing more than one characteristic, list each characteristic on a different line.)

2. Teaching

- [ ] (I) is a good tool for classroom demonstrations
- [ ] (J) supports real world examples
- [ ] (K) can be used for the duration of the course
- [ ] (L) supports a rigid framework where lessons and activities are easily incorporated and/or modified
- [ ] (M) Other (please specify: if listing more than one characteristic, list each characteristic on a different line.)

Pedagogical Tools Survey for Teachers

3. Which of the characteristics checked in the Learning and Teaching categories above do you perceive to be most significant for the EFFECTIVENESS of this tool.

Use the drop-down menus to choose the letter that corresponds to the characteristic in the list above. If possible, list your choices in order of importance with the most important characteristic listed first.

	Characteristics
1.	
2.	
3.	

(H or M) If your ranked H or M as 1, 2, or 3 and if you listed more than one characteristic for H or M, please use the space below to clarify the characteristic for the specified rank.

Pedagogical Tools Survey for Teachers

5. Materials and Tool Summary

The characteristics listed below relate to auxiliary materials and your perceptions.

Check the characteristics below that you believe contribute to the effectiveness of this tool in the teaching and learning of programming in an introductory programming course.

All characteristics are not applicable to all types of tools.

1. Auxiliary Materials

- ☐ (A) Technical support is provided by authors or community.
- ☐ (B) Tutorials on using the tool are provided by authors or community.
- ☐ (C) Instructor resources (PP presentations, sample problems and labs, syllabus) are provided by authors or community.
- ☐ (D) This tool can be used independent of the textbook chosen.
- ☐ (E) A mechanism for the sharing of materials is provided by authors or community.
- ☐ (F) Textbooks that incorporate this tool are available.
- ☐ (G) Materials about this tool can be found through on-line searches.
- ☐ (H) Other (please specify: if listing more than one characteristic, list each characteristic on a different line.)

2. Perceptions

- ☐ (I) After using this tool, your students are ready to extend their knowledge by continuing on in a computer science curriculum.
- ☐ (J) The initial experience with this tool positively affects subsequent programming experiences.
- ☐ (K) Using this tool eases and promotes the learning of programming.
- ☐ (L) Students enjoy using this tool.
- ☐ (M) Using this tool contributes to an increase in student enrollment in this course.
- ☐ (N) Using this tool contributes positively to student retention in course (students do not drop course).
- ☐ (O) Using this tool contributes positively to student retention in discipline (students take addition computer science courses).
- ☐ (P) Using this tool eases and promotes the teaching of programming.
- ☐ (Q) Other (please specify: if listing more than one characteristic, list each characteristic on a different line.)

Pedagogical Tools Survey for Teachers

3. Which of the characteristics checked in the Auxiliary Materials and Perceptions categories above do you perceive to be most significant for the effectiveness of this tool.

Use the drop-down menus to choose the letter that corresponds to the characteristic in the list above. If possible, list your choices in order of importance with the most important characteristic listed first.

	Characteristics
1.	
2.	
3.	

(H or Q) If your ranked H or Q as 1, 2, or 3 and if you listed more than one characteristic for H or Q, please use the space below to clarify the characteristic for the specified rank.

4. Which of the characteristics that you DID NOT check in the Auxiliary Materials would you like the tool to support (if any).

Use the drop-down menus to choose the letter that corresponds to the characteristic in the list above. If possible, list your choices in order of importance with the most important characteristic listed first.

	Characteristics
1.	
2.	
3.	

Other (please specify any characteristics NOT listed above that you would like this tool to support. Include your importance rank (1, 2, or 3))

Pedagogical Tools Survey for Teachers

6. Negatives

The characteristics listed below can be perceived as negative. Check those characteristics (if any) that you believe are NEGATIVE aspects of using this tool in that they either hinder your teaching or your students' learning.

1. Check the characteristics below that you feel are negative aspects of the TOOL ENVIRONMENT in that they either hinder your teaching or your students' learning. If the tool has no significant negative characteristics, check the first box.

- ☐ no significant negatives
- ☐ (A) The environment is too restrictive.
- ☐ (B) Students have technical difficulties.
- ☐ (C) The learning curve is too steep.
- ☐ (D) The tool is too big for introductory classes.
- ☐ (E) The tool is too restrictive to use for the duration of the course.
- ☐ (F) The transition to real-world programming is difficult.
- ☐ (G) The initial experience with tool negatively affects subsequent programming experience.
- ☐ (H) The ideas embodied in tool are shallow.
- ☐ (I) Good OO style is distorted by pragmatics and limitations of this tool.
- ☐ (J) Concepts learned with tool have to be unlearned in subsequent programming courses.

(K) Other (please specify: if listing more than one characteristic, list each characteristic on a different line.)

Pedagogical Tools Survey for Teachers

2. Check the characteristics below that you feel are negative aspects related to ERRORS AND SUPPORT in that they either hinder your teaching or your students' learning. If the tool has no significant negative characteristics, check the first box.

- ☐ no significant negatives
- ☐ (L) cryptic error messages
- ☐ (M) prevents syntax errors
- ☐ (N) does not prevent syntax errors
- ☐ (O) no easy way to debug
- ☐ (P) technical difficulties with installation
- ☐ (Q) no technical support
- ☐ (R) no instructor resources
- ☐ (S) no student resources
- ☐ (T) too costly
- ☐ (U) not multi-platform
- ☐ (V) not open source

(W) Other (please specify: if listing more than one characteristic, list each characteristic on a different line.)

Pedagogical Tools Survey for Teachers

7. Training and Experience

The questions on this page pertain to your training in the use of this tool and your experience using it in your courses.

1. What type of training in the use of this tool did you have before you introduced it in your programming course?

- [ ] (A) no training; knew nothing about tool; learned by experimentation
- [ ] (B) observed demonstration of tool at conference but no other training
- [ ] (C) 1-2 hour workshop introducing the tool
- [ ] (D) workshop lasting several days on using this tool
- [ ] (E) tutorial provided with tool
- [ ] (F) users guide or manual provided with this tool
- [ ] (G) colleague provided information needed to use this tool
- [ ] (H) Other (please specify)

2. Please indicate your agreement with each of the following statements.

	strongly agree	agree	neither agree nor disagree	disagree	strongly disagree
The amount of training I received in the use of this tool was enough.	○	○	○	○	○
I will continue to use the tool in future courses.	○	○	○	○	○
I will recommend this tool to other computer science educators.	○	○	○	○	○
Using this tool in my introductory programming class eases and promotes the LEARNING of programming concepts.	○	○	○	○	○
Using this tool in my programming class eases and promotes the TEACHING of programming concepts.	○	○	○	○	○
Using this tool in my introductory programming class eases and promotes the LEARNING of OO concepts.	○	○	○	○	○
Using this tool in my programming class eases and promotes the TEACHING of OO concepts.	○	○	○	○	○

Pedagogical Tools Survey for Teachers

3. If you have used other pedagogical tools in your course, rank the effectiveness of this tool compared to others you have used.

	Most effective tool that I've used	More effective than most tools	About the same as most tools	Less effective than most tools	Least effective tool that I've used	No other tools used
effectiveness of this tool as compared to other tools	○	○	○	○	○	○

4. Please add any comments about this tool that you would like to share.

5. If you have any knowledge of research done (or in progress) that involves measured effectiveness of this tool and you can provide references, please list the references below.

6. In addition to the this tool, what pedagogical tools (if any) do you use in this course? If time permits, please complete this survey for each tool.

	Tool
1.	
2.	
3.	
4.	
5.	

Other: Please list each tool on a different line.

Pedagogical Tools Survey for Teachers

7. What pedagogical tools (if any) have you used in your programming courses that you no longer use?

The information about tools consciously NOT chosen is very valuable. If time permits, please complete this survey for each tool you chose NOT to use.

	Tool
1.	
2.	
3.	
4.	
5.	

Other: Please list each tool on a different line.

8. Please add any additional comments that you would like to share.

Pedagogical Tools Survey for Teachers

8. Your course

The questions below will provide information about the course in which you use this tool.

1. What is the name of your course?

2. Please enter the approximate length of your course (in hours).

Total Hours of class meetings

3. Check the answer that best describes the status of your course.

- [ ] elective
- [ ] general education requirement
- [ ] majors requirement
- [ ] honors course
- [ ] AP course
- [ ] Other (please specify)

4. How many times have you taught this course?

- ○ 1
- ○ 2
- ○ 3
- ○ 4
- ○ 5
- ○ more than 5

5. In how many classes have you used this tool?

- ○ 1
- ○ 2
- ○ 3
- ○ 4
- ○ 5
- ○ more than 5

Pedagogical Tools Survey for Teachers

9. About You

The questions below provide informatioon about you.

1. Your gender

- ○ Male
- ○ Female

2. Your affiliation

- ☐ middle school
- ☐ secondary school
- ☐ 2-year college
- ☐ 4-year college
- ☐ other

other (please specify)

Pedagogical Tools Survey for Teachers

10. More About You

The following information will be kept private. You will not be identified in any publication or presentation of the study findings.

1. Your Name

2. Your Affiliation

3. Your email address

4. If you have used at least five of these tools (or chose NOT to use them) in your courses, would you be willing to help me by participating in an informal interview?

- yes
- no

Pedagogical Tools Survey for Teachers

11. Thank You

Thank you for completing this survey!
You will now be redirected to the first page of the survey with hopes that you might complete this for another tool.
To exit, simply close the window.

Email me if you have any questions.

Fran Trees
ftrees@drew.edu

1. Please indicate below if you would like to be informed when a summary of the survey results are available.

- ○ Yes, please send me a link where I can view the results.
- ○ No, thank you.

Appendix B

Survey Letters

B.1 Survey Notication

Hi!!!

I would like to warn you that I am about to ask you a BIG favor! I will be sending you information on completing a survey on pedagogical tools. An introduction about the survey will be in my next email to you. I am working on this for my dissertation and I would REALLY appreciate your responses. I know that most of you have used tools in your teaching. If you do NOT use tools, I would appreciate knowing why you choose NOT to use tools.

My survey is based on introductory programming classesthis could be CS0, CS1 in universities, pre-AP, AP or other intro courses in high schools and intro to computing in middle schools.

You may not be teaching these courses now but you are on my list because I value your opinions. If you have ever taught using tools, please complete the survey. Thanks in advance for your help!

Fran

Frances P. Trees

Drew University

Math/ CS Department

ftrees@drew.edu

B.2 Survey Invitation

Please help me!!!

I am conducting research for my dissertation on pedagogical tools used by computer science educators in introductory programming courses. These tools can be categorized in a variety of ways: Algorithm Visualizations, Algorithm Animations, Microworlds, Integrated Development Environments, Robots, Game Making Tools, etc.

I have tried many of them in my own classes, some successfully and some not. In an effort to identify the characteristics of an effective pedagogical tool for an introductory programming course, I am requesting that you complete a survey for the tool (or tools) that you use (or have used) in your programming course. I am interested in both your successful experiences and your unsuccessful experiences!

Some of the popular tools are listed below but there are hundreds of others.

Alice, BlueJ, Buggles and Bagels, CodeWiz, DrJava, DrScheme, Eclipse, GameMaker, Greenfoot, Gridworld, Greeps, JamTester, Java Task Force Graphics, Jeliot 3, Jeroo, jGrasp, JHAVE, JPie, Java Power Tools, Junit, Karel J. Robot, LEGO Mindstorms, Media Computation, NetBeans, ObjectDraw, PigWorld, RAPTOR, Robotran, Scratch, XTANGO, ZEUS

The nature of my research is focused the teacher's perception of tool effectiveness and not measured effectiveness of student outcomes. What characteristics do you (the teacher) see as contributing to the effectiveness of the tool? In other words, why do you choose to use some tools and choose not to use others?

The survey that I am asking you to complete is divided into several "pages" or parts:

1. General information (what tool are you using, why did you choose this tool)
2. General tool characteristics that you feel contribute its effectiveness.
3. Programming environment, testing, debugging, and interaction characteristics that you feel contribute its effectiveness.
4. Characteristics that contribute to the effectiveness of this tool as they relate to

student learning and your teaching.

5. Auxiliary materials that contribute to the effectiveness of the tool and your perceptions of using the tool in your class.

6. Page 6 focuses on NEGATIVE aspects of the tool (if you feel any exist).

7. Page 7 gathers information about the amount of training you had and your experience using the tool.

8. Pages 8, 9, and 10 collect basic information on your course and you.

Yes, it does seem long but the questions are objective: check characteristics that contribute the tool's effectiveness and choose the most important of these characteristics. Most pages have only 2-4 questions. The survey should take about 15 minutes to complete. All questions are not applicable for all tools. If the question is not applicable, check "does not apply." Your opinions are important to me. The tools you choose NOT to use and the reasons you chose NOT to use them are as important as the tools you continue to use. Please help me by completing this survey for at least one tool that you value (preferably more) and one tool that you have tried and are no longer using.
The survey is available at:

`http://www.surveymonkey.com/s.aspx?sm=OQkR_2fsS1DsgdnFiiLgKgLA_3d_3d`

If possible, please complete the survey by *date*.
Thank you SO much!
Fran Trees
ftrees@drew.edu

B.3 Invitation Follow-up

Hello again!

I know you are probably very busy and have more than enough to fill your free moments (if you have any) but I don't want you to forget about me (actually my survey)! If you haven't already completed my survey about pedagogical tools used in introductory programming courses, please remember to do so by *date*! I do appreciate your help! The survey is available at:

`http://www.surveymonkey.com/s.aspx?sm=OQkR_2fsS1DsgdnFiiLgKgLA_3d_3d`

Feel free to share this URL with a colleague! If you have any questions, please let me know!

Thanks again.

Fran

Frances P. Trees

Drew University

Math/ CS Department

ftrees@drew.edu

Appendix C

Interview Questions

Interview Questions for Teachers

You indicated that you use (or consciously chose NOT to use) 5 or more of the tools listed. You've used more tools in your class than many teachers do. Actually, some teachers choose not to use any tools.

1. What tools have you used?
2. How do you find appropriate tools to use?
3. What influences you to use a particular tool?
4. Some research indicates that teachers do not use tools in their class because
    a. They do not have the time to spend learning the tool.
    b. They cannot afford the time to use the tool in their class because it would take time away from the curriculum that they MUST cover.

    How do you respond to these issues? (How do you find the time to learn the tools?)

    What were your first-time experiences using this tool?
5. If you had the chance to work with the author(s) of the tools that you use (or have used) what experiences and information would you share what them?
6. If you really believe in the effectiveness of a tool and you wanted to tell the world of introductory programming teachers about the tool, what mechanisms would you use to communicate with these teachers? (Note: all introductory programming teachers ARE NOT hooked into listserves.)
7. Do you use any tool strictly for classroom demonstrations?
    a. If so, why is this tool restricted to demonstrations?
8. Of the tools that you use, do you believe that you use the tool to its full potential?
    a. If not, why not?
        i. What prevents you from using the other aspects of the tool?
    b. What percent of the tool's capabilities do you use?
9. Many of the tools used are used by teachers who learn the tool on the fly.
    a. What are your thoughts on that?
    b. What is the best medium to train in the use of a tool?
    c. (*leading)* Would you participate in a Web training event for the tools that you use
        i. as a student (participant)
        ii. as a support person (presenter)
10. Would your teaching change is the tool were taken away?
    a. if so, how?
11. *(purposely leading)* What tools do we need that aren't out there?
    a. What should they be like?
12. Are there any tools that you consciously choose NOT to use?
    a. What are they?
    b. Have you tried them?

c. Why don't you use them?

13. You've used many tools.... If you could choose only one tool to keep using, what would it be?

a. Why?

14. What did I forget to ask?

Appendix D

Mapping of Developer's Goals with Tool Characteristics

Developer's Goals	Survey Questions
easy to use[200, 157] reduces the complexity of details [50] students can start using the environment on their own almost immediately[124]	has an intuitive interface and has a learning curve that is not too steep
provides object interaction and inspection [124] supports user interaction programming via direct state manipulation [123]	supports an interactive environment
engages students immediately [200] actively engage students in increasing knowledge and skills [49] facilitate a more engaging, less frustrating first programming experience[6]	supports students active engagement in learning activities improves first-time programmer's experience encourages learning through discovery
provides students with a better understanding of programming concepts[200]	supports an understanding of abstract concepts helps understanding of program execution

continued on next page

continued from previous page

Developer's Goals	Survey Questions
increases student satisfaction and enthusiasm for programming[200]	students enjoy using this tool the initial experience of this tool positively affects subsequent programming courses
consistent visualization theme[157] we depend on metaphor to help us teach [21] a kangaroo-like animal living on Santong Island is programmed [200]	supports consistent metaphor
improves comprehensibility of software and provides support for understanding objects representing data structures [54, 109]	supports understanding of abstract and complex concepts
shows program structure[124] goal is to develop crucial programming skills[60]	embodies ideas that support core programming concepts supports visual representation of general programming concepts
allows students to instantiate objects and explore inheritance[50, 54, 157] supports objects first approach [50, 124] provides clean illustration of OO concepts [123] introduces dynamic polymorphism [22] guided by the prospect of an "objects-first" approach [104]	supports understanding of OO concepts supports visualization of OO concepts
provides a sense of program state[50] provides direct state manipulation and object inspection [123]	supports visualization of program state supports visualization of state changes

continued on next page

continued from previous page

Developer's Goals	Survey Questions
visualize the steps of the execution of the program [49, 107]	supports visualization of program code
no correlation between performance on the visual questions and the preferred learning style [84] allows teachers to address students in locally or personally relevant ways[96] allows to vary the complexity and thus the difficulty level of the material to be learned [96] environment supports the whole spectrum of programmers - from complete novices to world class researchers and real world applications[178]	allows use at many levels of learning engages students of different ability levels allows students to work at levels of their choice
possible to intersperse tool throughout the semester [50] design goal is to support programming in the first year [124]	can be used for the duration of the course
students gain an understanding of the coordinate system and the spatial relationship [49] example problems have a graphical user interface[124] can be used for a wide variety of graphical applications [123]	includes graphical components
supports highly flexible scenarios [123]	does not restrict scenarios
students write programs to control one or more robots in a rectangular	restricts scenarios

continued on next page

continued from previous page

Developer's Goals	Survey Questions
world [21] A Jeroo is a rare kangaroo-like animal living on Santong Island [60]	
easy to create an animation for telling a story[6]	incorporates storytelling
beneficial to explicitly address the transition to the next environment in discussions[124] supports migration to other environments [123]	allows for easy transition to a more robust environment
reduces the complexity of details [50] gently introduces students to the mechanics of writing Java programs[7] provides GUI components appropriate for beginners [104]	simplifies mechanics of programming
permits very complex programs to be written with a very reduced Java subset [22]	sophisticated problems can be solved
a 3D interactive, animation, programming environment for building virtual worlds[6]	media rich programming environment
provide animated execution and code highlighting [200]	supports step by step evaluation of programming statements
enables the user to explore accessibility and visibility relationships by experimenting with any object on the workbench interactive object calls, interactive testing, and incremental development [28] support development of debugging skills [50]	supports developing and testing individual components supports debugging provides meaningful error messages

continued on next page

continued from previous page

Developer's Goals	Survey Questions
error messages are sometimes cryptic[49] includes a source level debugger [123]	
highly visual feedback [49] provide visual feedback of object state and behaviour [123]	provides immediate feedback
students do not develop a detailed sense of syntax [50] "no-typing" syntax-automated editor[49] detect basic syntactic errors as soon as possible [7]	addresses (or not) syntax issues
information in tutorials provided by developers [6, 28]	supports comments and documentation
information in tutorials provided by developers [6, 28]	supports user provided input structures
information in tutorials provided by developers [6, 107, 24] the Data Structure Identifier determines what type of object and opens the appropriate viewer	supports introduction of data
the group work aspect is not a coincidental side issue [124]	provides natural platform for encouraging group work
visualization is a superior was to visualize behavior [157]	good tool for classroom demonstration
introduces problem solving approaches that can be used with computers [25] draw on relevant examples and uses in their field	supports real world examples

continued on next page

continued from previous page

Developer's Goals	Survey Questions
and that emphasize computing concepts and skills that go beyond just software development [89] powerful enough that it scales to testing of large systems[103]	
provides a framework and environment [123]	supports rigid framework where lessons easily incorporated
reduce the attrition of our most at-risk majors [50] increase in percentage of students who continued on to CS2 [50] We are enjoying dramatically higher retention rates[89]	contributes to increase in enrollment contributes to increase in student retention in course contributes to increase in student retention in discipline students ready to continue studying CS
information on web site [6]	technical support is provided
information on web site[6, 28, 77, 104]	tutorials provided
information on web site[6, 24]	instructor resources provided
information on web site[6, 28, 77, 24, 152]	textbooks that incorporate this tool are available
information on web site[6, 77, 152]	a mechanism for community sharing of materials is provided
information on web site[6, 28, 61] [64, 77, 82, 107, 108] [109, 103, 104, 24, 152, 166]	free, multi platform, and/or open source

Table 45: Mapping of Developers' Goals with Tool Characteristics from Survey Questions

Appendix E

Themes

Theme	Characteristic	Micro Worlds	Libraries	IDEs	Visualization Tools	Total	Male	Female	College	HS	Users	Non Users
Tool Choice	Recommended by CS educators	62.3	66.7	63.6		60.8	54	70.4		69.3	57.1	78.4
	Task-technology fit				57.1				54.4			
	Compliments teaching style								52.6			
Manageable Environment	Has an intuitive interface	63.4			84.6	62.6	63	63.3	57.9	64.8	64.5	51.5
	Has a learning curve that is not steep	74.6		74.8	100	74.4	75	75.9	77.2	74.4	76.9	60.6
	Tool is not too big and not too restrictive											
	No technical difficulties with installation											
	Simplifies the mechanics of programming	52.9		55.4	61.5	53.5	59		63.2		53.5	50
Active Learning	Supports an interactive environment	85.9	83.3	65.4	84.6	71.9	72	73.4	71.9	73.6	74	60.6
	Supports students' active engagement in learning activities	93.8	100	60.9	84.6	72.8	67.7	80	65.8	76.7	73	70.4
	Encourages learning through discovery	79.7				53.3		58.7	50	54.2	52	63
Good First Experience	Improves first-time experience	84.5	100	72	84.6	75.4	75	75.9	70.2	78.4	78.1	60.6
	Students enjoy using this tool	93.1	100	67.4	70	77.2	70.2	85.9	64.2	83.5	77.6	78.3
	Introduces programming in a more enjoyable way	89.6	100	50	57.1	65.8	65	70.4	66.7	67.6	63	78.4
	The initial experience positively affects subsequent programming experiences	70.7	100	60.7		65.3	66	63.4	60.4	67.8	67.8	

contribute significantly

inadequately supported

contribute significantly but are still inadequately supported

Theme	Characteristic	Micro Worlds	Libraries	IDEs	Visualization Tools	Total	Male	Female	College	HS	Users	Non Users
Visual Environment	Includes graphical components	84.5	83.3			54.2	51	57		58.4	52.7	63.6
	Supports visualization of OO concepts	58.8		50.5		50.1		53.9		53.3	52.2	
	Provides multiple views at once	54.4		55.4	53.8					51.6		
	Supports visualization of program state	64.7			53.8							
	Supports visualization of state changes	54.4			53.8							
	Supports visualization of program code											
	Provides a media rich environment											
	Supports visual representation of general programming concepts											
Flexible Environment	Allows for use at many levels of learning	59.2	83.3	58.9	76.9	59.1	63	59.5	50.9	65.6	62.7	
	Can be used for the duration of the course		83.3	87.5	90.9	68.4	80.2	73.6	76.4	66.4	74	
	Attracts a diverse group of students	66.2										
	Engages students of different ability levels	81.3	83.3	51.1		64.4	60.4	69.3	55.6	69.2	67.8	
	Allows students to work at levels of their choice	60	66.7					50.7		50.8		

contribute significantly
inadequately supported
contribute significantly but are still inadequately supported

Theme	Characteristic	Micro Worlds	Libraries	IDEs	Visualization Tools	Total	Male	Female	College	HS	Users	Non Users
Subsequent Courses	Allows for an easy transition to a more robust programming environment											
	Good OO style is NOT distorted by pragmatics and limitations of this tool											
	Transition to real world is NOT difficult											
	After using this tool students are ready to extend their knowledge by continuing in CS curricula	65.5		79.8		72.5	76.6	66.2	64.2	76.5	74.8	56.5
	Helps with retention in course											
Tool Resources	Mechanism for sharing materials is provided by author or community											
	Instructor Resources are provided by author or community	68.3	100							52.8	51.3	
	Technical support is provided by authors or community	68.3	100	62.1		63	67.7	56.3	57.9	65.6	62.7	63.3
	Tool can be used regardless of textbook chosen	81	100	90.5	100	84.5	84.8	83.8	93	80.8	89.3	60
	Tutorials or Teachers Guide provided by authors or community	66.7		67.4	63.6	63.5	65.7	61.3	66.7	62.4	63.3	63.3
	Materials can be found through on line search	82.5	66.7	68.4		70.2	68.7	72.5	70.2	71.2	69.3	73.3
	Textbooks that use the tool are available	61.9	83.3	50.5		54.7	53.5	55	52.6	55.2	53.3	60

contribute significantly
inadequately supported
contribute significantly but are still inadequately supported

Theme	Characteristic	Micro Worlds	Libraries	IDEs	Visualization Tools	Total	Male	Female	College	HS	Users	Non Users
Programming	Supports step-by-step evaluation of single programming statements			60.4	84.6	50.3	56.8		53.7		50.3	
	Supports incremental development											
	Supports comments and documentation			70.3	53.8	54.1	56.8	51.4	57.4	53.8	59.2	
	Gives immediate feedback about errors			57.4	53.8	50.3	51.6			50.4	52.2	
	Supports event-driven programming	50.8										
	Supports testing of individual components			55.4		51.9	60		68.5		52.9	
	Supports debugging in an easy way			74.3	84.6	53.6	56.8		61.1	50.4	57.3	
	Prevents syntax errors	52.5										
	Provides meaningful error messages (no cryptic error messages)							53				
	supports introduction of data structures											
Teaching	Eases and promotes the teaching of programming	81	83.3	68.5	70	73.1	72.3	73.2	67.9	75.7	76.2	52.2
	Good tool for class demonstrations	91.9	83.3	77.1	90.9	80.8	80.2	80.6	72.7	84.5	82	73.1

contribute significantly

inadequately supported

contribute significantly but are still inadequately supported

Theme	Characteristic	Micro Worlds	Libraries	IDEs	Visualization Tools	Total	Male	Female	College	HS	Users	Non Users
Learning	Eases and promotes the learning of programming	79.3	100	76.4	90	76.6	71.3	83.1	69.8	80	81.8	
	Helps understanding of program execution	65.5	66.7	57.6	60	60.6	58.3	62.7	57.4	61.7	63.8	
	Supports the understanding of abstract and complex concepts	64.1	66.7	52.2		54.4	56.3	50.7	50	56.7	55.9	
	Supports abstraction											
	Embodies ideas that support core programming concepts	69	83.3			57.1	60	55.7	54.4	60.8	59.2	
	Supports a concepts first approach	67.6										
	Supports consistent metaphor	59.2										

contribute significantly

inadequately supported

contribute significantly but are still inadequately supported

References

[1] Accentance, Inc. Website. `http://www.accentance.com/`.

[2] ACM SIGCSE Committee. Engaging computer science education. In *SIGCSE '09: Proceedings of the 40th SIGCSE technical symposium on Computer science education*, New York, NY, USA, 2009. ACM.

[3] Joel Adams and Jeremy Frens. Object centered design for Java: Teaching OOD in CS-1. *SIGCSE Bull.*, 35(1):273–277, 2003.

[4] Mohammed Al-Bow, Debra Austin, Jeffrey Edgington, Rafael Fajardo, Joshua Fishburn, Carlos Lara, Scott Leutenegger, and Susan Meyer. Using Greenfoot and games to teach rising 9th and 10th grade novice programmers. In *Sandbox '08: Proceedings of the 2008 ACM SIGGRAPH symposium on Video games*, pages 55–59, New York, NY, USA, 2008. ACM.

[5] Kirsti Ala-Mutka. Problems in learning and teaching programming - a literature study for developing visualizations in the Codewitz-Minerva project. `http://www.cs.tut.fi/~edge/literature_study.pdf`.

[6] Alice Website, cited April 2009. `http://www.Alice.org`.

[7] Eric Allen, Robert Cartwright, and Brian Stoler. DrJava: a lightweight pedagogic environment for Java. *SIGCSE Bull.*, 34(1):137–141, 2002.

[8] James Allert. Learning style and factors contributing to success in an introductory computer science course. In *ICALT '04: Proceedings of the IEEE International Conference on Advanced Learning Technologies*, pages 385–389, Washington, DC, USA, 2004. IEEE Computer Society.

[9] Dorine Andrews, Blair Nonnecke, and Jennifer Preece. Conducting Research on the Internet: Online survey design, development, and implementation guidelines. *International Journal of Human-Computer Interaction*, 16(2):185–210, 2003.

[10] Karen Anewalt. Making CS0 fun: an active learning approach using toys, games and Alice. *J. Comput. Small Coll.*, 23(3):98–105, 2008.

[11] AP Computer Science Electonic Discussion Group. `http://lyris.collegeboard.com/read/?forum=ap-compsci`.

[12] Owen Astrachan and Susan H. Rodger. Animation, visualization, and interaction in CS 1 assignments. In *SIGCSE '98: Proceedings of the twenty-ninth SIGCSE technical symposium on Computer science education*, pages 317–321, New York, NY, USA, 1998. ACM.

[13] Frances Bailie, Glenn Blank, Keitha Murray, and Rathika Rajaravivarma. Java visualization using BlueJ. *J. Comput. Small Coll.*, 18(3):175–176, 2003.

[14] Jessica D. Bayliss. Using games in introductory courses: tips from the trenches. In *SIGCSE '09: Proceedings of the 40th ACM technical symposium on Computer science education*, pages 337–341, New York, NY, USA, 2009. ACM.

[15] Theresa Beaubouef and John Mason. Why the high attrition rate for computer science students: some thoughts and observations. *SIGCSE Bull.*, 37(2):103–106, 2005.

[16] Byron Weber Becker. Teaching CS1 with Karel the Robot in Java. *SIGCSE Bull.*, 33(1):50–54, 2001.

[17] Henry Jay Becker and Margaret M. Riel. Teacher professionalism and the emergence of constructuvust-compatible pedagogies, 1999. Revised version of a paper presented at the 1999 meeting of the American Educational Research Association, Montreal.

[18] Mordechai Ben-Ari. Constructivism in computer science education. In *SIGCSE '98: Proceedings of the twenty-ninth SIGCSE technical symposium on Computer science education*, pages 257–261, New York, NY, USA, 1998. ACM.

[19] Jens Bennedsen and Michael E. Caspersen. An investigation of potential success factors for an introductory model-driven programming course. In *ICER 05: Proceedings of the 2005 international workshop on Computing education research*, 2005.

[20] Jens Bennedsen and Michael E. Caspersen. Abstraction ability as an indicator of success for learning object-oriented programming? *ACM SIGCSE Bulletin*, 38(2):39–43, 2006.

[21] Joe Bergin. Karel Universe Drag & Drop Editor. In *ITICSE '06: Proceedings of the 11th annual SIGCSE conference on Innovation and technology in computer science education*, pages 307–307, New York, NY, USA, 2006. ACM.

[22] Joe Bergin, Kim Bruce, and Michael Kölling. Objects-early tools: a demonstration. *SIGCSE Bull.*, 37(1):390–391, 2005.

[23] Joe Bergin, Raymond Lister, Barbara Boucher Owens, and Myles McNally. The first programming course: ideas to end the enrollment decline. *SIGCSE Bull.*, 38(3):301–302, 2006.

[24] Joe Bergin, Mark Stehlik, Jim Roberts, and Richard Pattis. Karel J. Robot website, cited April 2009. `http://csis.pace.edu/~bergin/KarelJava2ed/Karel\%2B\%2BJavaEdition.html`.

[25] Joseph Bergin, Mark Stehlik, Jim Roberts, and Rich Pattis. *Karel J Robot: A Gentle Introduction to the Art of Object-Oriented Programming in Java*. Dream Songs Press, 2005.

[26] Susan Bergin and Ronan Reilly. Programming: Factors that influence success. In *Proceedings of the 32nd SIGCSE Technical Symposium on Computer Science Education*, 2005.

[27] Betsy Bizot. US CS new majors, enrollment both rise in 2007-2008. *Computing Research News*, 21(2), 2009.

[28] BlueJ, cited April 2009. `http://www.bluej.org/`.

[29] Marcella Bombardieri. In computer science, a growing gender gap: Women shunning a field once seen as welcoming. Boston Globe, 18 December 2005, 2005.

[30] Tom Briggs. Techniques for active learning in CS courses. *J. Comput. Small Coll.*, 21(2):156–165, 2005.

[31] Kim Bruce, Andrea Danyluk, and Tom Murtagh. Java: An Eventful Approach. *J. Comput. Small Coll.*, 19(5):64–65, 2004.

[32] Kim B. Bruce. Controversy on how to teach CS 1: a discussion on the SIGCSE-members mailing list. *SIGCSE Bull.*, 37(2):111–117, 2005.

[33] Kim B. Bruce, Andrea Danyluk, and Thomas Murtagh. A library to support a graphics-based object-first approach to CS 1. In *SIGCSE '01: Proceedings of the thirty-second SIGCSE technical symposium on Computer Science Education*, pages 6–10, New York, NY, USA, 2001. ACM.

[34] Pat Byrne and Gerry Lyons. The effect of student attributes on success in programming. *SIGCSE Bull.*, 33(3):49–52, 2001.

[35] Wayne J. Camara and Roger Millsap. Using the PSAT/NMSQT and course grades in predicting success in the Advanced Placement Program. In *College Board Report No. 98-4*. The College Board, 1998.

[36] Yan Cao, Zhou Fang, Yanli Yang, and Zhong Li. Cutter Database Management System Development on NetBeans 5.0 Platform. *First International Workshop on Database Technology and Applications, 2009*, pages 41–44, 2009.

[37] Angela Carbone, Linda Mannila, and Sue Fitzgerald. Computer science and IT teachers' conceptions of successful and unsuccessful teaching: A phenomenographic study. *Computer Science Education*, 17(4):275–299, 2007.

[38] Martin C. Carlisle, Terry A. Wilson, Jeffrey W. Humphries, and Steven M. Hadfield. RAPTOR: a visual programming environment for teaching algorithmic problem solving. In *SIGCSE '05: Proceedings of the 36th SIGCSE technical symposium on Computer science education*, pages 176–180, New York, NY, USA, 2005. ACM.

[39] Michael E. Caspersen. *Educating Novices in The Skills of Programming*. PhD thesis, University of Aarhus, 2007.

[40] A. T. Chamillard. Introductory game creation: no programming required. In *SIGCSE '06: Proceedings of the 37th SIGCSE technical symposium on Computer science education*, pages 515–519, New York, NY, USA, 2006. ACM.

[41] Zhixiong Chen and Delia Marx. Experiences with Eclipse IDE in programming courses. *J. Comput. Small Coll.*, 21(2):104–112, 2005.

[42] Arthur Chickering and Stephen C. Ehrmann. Seven principles of good practice in undergraduate education. *International Journal of Human-Computer Interaction*, 39:3–7, 1987.

[43] Hyunyi Cho and Robert Larose. Privacy issues in Internet surveys. *Social Science Computer Review*, 17(4):421, 1999.

[44] Donald R. Clark. Kolb's Learning Styles and Experimental Learning Model, cited July 2009. `http://www.nwlink.com/~donclark/hrd/styles/kolb.html`.

[45] Daniel C. Cliburn. The effectiveness of games as assignments in an introductory programming course. In *Proceedings of the 36th ASEE/IEEE Frontiers in Education Conference*, San Diego, CA, USA, 2006.

[46] Louis Cohen, Lawrence Manion, and Keith Morrison. *Research Methods in Education*. Routledge, New York,NY, USA, 2007.

[47] J. McGrath Cohoon and Lih-Yuan Chen. Migrating out of computer science. *Computing Research News*, 15(2), 2003.

[48] Commonwealth of Learning. *Manual for Educational Media Researchers: Knowing your Audience*, page Chapter 13. The Commonwealth of Learning, Commonwealth Educational Media Centre of Asia, cited December 2009.

[49] Stephen Cooper, Wanda Dann, and Randy Pausch. Alice: a 3-D tool for introductory programming concepts. In *CCSC '00: Proceedings of the fifth annual CCSC northeastern conference on The journal of computing in small colleges*, pages 107–116, , USA, 2000. Consortium for Computing Sciences in Colleges.

[50] Stephen Cooper, Wanda Dann, and Randy Pausch. Teaching objects-first in introductory computer science. In *SIGCSE '03: Proceedings of the 34th SIGCSE technical symposium on Computer science education*, pages 191–195, New York, NY, USA, 2003. ACM.

[51] John W. Cresswell, William E. Hanson, Vicki L Clark Plano, and Alejandro Morales. Qualitative research designs: Selection and implementation. *The Counseling Psychologist*, 35:236–264, 2007.

[52] James H. Cross, II. jGRASP: teaching hard concepts with intuitive visualizations: conference workshop. *J. Comput. Small Coll.*, 24(1):254–256, 2008.

[53] James H. Cross, II and T. Dean Hendrix. jGRASP: an Integrated Development Environment with visualizations for teaching Java in CS1, CS2, and beyond. *J. Comput. Small Coll.*, 23(3):169–171, 2008.

[54] James H. Cross, II, T. Dean Hendrix, Jhilmil Jain, and Larry A. Barowski. Dynamic object viewers for data structures. In *SIGCSE '07: Proceedings of the 38th SIGCSE technical symposium on Computer science education*, pages 4–8, New York, NY, USA, 2007. ACM.

[55] CSTA. Ensuring exemplary teaching in an essential discipline: Addressing the crisis in computer science teacher certification. Technical report, CSTA, 2008.

[56] CSTA. CS & IT Symposia, cited in December 2009. `http://www.csta.acm.org/ProfessionalDevelopment/sub/CSITSymposiaSite.html`.

[57] Perry Davis, Kimberly Yonce, and Caroline Eastman. The impact of using robotics on student program complexity in CS1 and CS2. Work conducted during a Research Experience for Undergraduates Summer 2007 program supported by NSF grant 0649105.

[58] Norman K. Denzin and Yvonna S. Lincoln. *Handbook of Qualitative Research*. Sage publications, Newbury Park, CA, 2nd edition, 2000.

[59] Dwight Deugo. Using Eclipse in the classroom. In *ITiCSE '08: Proceedings of the 13th annual conference on Innovation and technology in computer science education*, pages 322–322, New York, NY, USA, 2008. ACM.

[60] Brian Dorn and Dean Sanders. Using Jeroo to introduce object-oriented programming. In *FIE '03: Proceedings of the 33rd annual Frontiers in Education Conference*, 2003.

[61] DrJava, cited April 2009. `http://www.drjava.org`.

[62] DrScheme, cited December 2009. `http://www.plt-scheme.org/`.

[63] Clive L. Dym, William H. Wood, and Michael J. Scott. Rank ordering engineering designs: pairwise comparison charts and Borda counts. *Research in Engineering Design*, 13(4):236–242, 2002.

[64] Eclipse, cited April 2009. `http://www.eclipse.org/`.

[65] Barry Fagin and Laurence Merkle. Measuring the effectiveness of robots in teaching computer science. *SIGCSE Bull.*, 35(1):307–311, 2003.

[66] Barry S. Fagin and Laurence Merkle. Quantitative analysis of the effects of robots on introductory computer science education. *J. Educ. Resour. Comput.*, 2(4):2, 2002.

[67] Richard M. Fedler. Reaching the second tier: Learning and teaching styles in college education. *Journal of College Science Teaching*, 23(5):286–290, 1993.

[68] James B. Fenwick, Jr., Cindy Norris, Frank E. Barry, Josh Rountree, Cole J. Spicer, and Scott D. Cheek. Another look at the behaviors of novice programmers. In *SIGCSE '09: Proceedings of the 40th ACM technical symposium on Computer science education*, pages 296–300, New York, NY, USA, 2009. ACM.

[69] Stefan Feyock and Thomas Ford. Individual learning styles and computer science education. In *ACM '76: Proceedings of the annual conference*, pages 130–134, New York, NY, USA, 1976. ACM.

[70] Sally Fincher and Marian Petre, editors. *Computer Science Education Research*, chapter 3. Routledge, 2004.

[71] Kasper Fisker, Davin McCall, Michael Kölling, and Bruce Quig. Group work support for the BlueJ IDE. In *ITiCSE '08: Proceedings of the 13th annual conference on Innovation and technology in computer science education*, pages 163–168, New York, NY, USA, 2008. ACM.

[72] Consortium for Computing Sciences in Colleges. CCSC website, cited in December 2009. `http://www.ccsc.org/`.

[73] International Society for Technology in Education. National Educational Computing Conference, cited in December 2009. `http://center.uoregon.edu/ISTE/NECC2009/program/`.

[74] Andrea Forte and Mark Guzdial. Computers for communication, not calculation: Media as a motivation and context for learning. *Hawaii International Conference on System Sciences*, 4:40096a, 2004.

[75] Carol Frieze. *The Critical Role of Culture and Environment as Determinants of Women's Participation in Computer Science*. PhD thesis, Carnegie Mellon University, 2007.

[76] Vashti C. Galpin, Ian D. Sanders, and Pei-yu Chen. Learning styles and personality types of computer science students at a South African University. In *ITiCSE '07: Proceedings of the 12th annual SIGCSE conference on Innovation and technology in computer science education*, pages 201–205, New York, NY, USA, 2007. ACM.

[77] Game Maker, cited April 2009. `http://www.yoyogames.com/gamemaker/`.

[78] Dan Garcia. Technology that Educators of Computing Hail. `http://www.cs.berkeley.edu/~ddgarcia/papers/SIGCSE2010TECHBOF.pdf`.

[79] Ray Giguette. Pre-games: games designed to introduce CS1 and CS2 programming assignments. *SIGCSE Bull.*, 35(1):288–292, 2003.

[80] Annagret Goold and Russell Rimmer. Factors affecting performance in first-year computing. *SIGCSE Bull.*, 32(2):39–43, 2000.

[81] Gina Green. Perceived control of software developers and its impact on the successful diffusion of information technology, 1999. Special Report: CMU/SEI-98-SR-013.

[82] Greenfoot, cited April 2009. `http://www.greenfoot.org/index.html`.

[83] Gridworld, cited April 2009. `http://apcentral.collegeboard.com/apc/public/courses/teachers_corner/151155.html`.

[84] Scott Grissom, Myles F. McNally, and Tom Naps. Algorithm visualization in CS education: comparing levels of student engagement. In *SoftVis '03: Proceedings of the 2003 ACM symposium on Software visualization*, pages 87–94, New York, NY, USA, 2003. ACM.

[85] Paul Gross and Kris Powers. Evaluating assessments of novice programming environments. In *ICER '05: Proceedings of the first international workshop on Computing education research*, pages 99–110, New York, NY, USA, 2005. ACM.

[86] Paul Gross and Kris Powers. Work in progress - a meta-study of software tools for introductory programming. In *Frontiers in Education, 2005. FIE '05. Proceedings 35th Annual Conference*, pages S1E–12, Indianopolis, IN, USA, 2005.

[87] Mario Guimaraes and Meg Murray. An exploratory overview of teaching computer game development. *J. Comput. Small Coll.*, 24(1):144–149, 2008.

[88] Mark Guzdial. A media computation course for non-majors. In *ITiCSE '03: Proceedings of the 8th annual conference on Innovation and technology in computer science education*, pages 104–108, New York, NY, USA, 2003. ACM.

[89] Mark Guzdial. Computing for everyone: Improving global competitiveness and understanding of the world. A White Paper for ICER Workshop, 2006.

[90] Mark Guzdial. Education: Teaching computing to everyone. *Commun. ACM*, 52(5):31–33, 2009.

[91] Mark Guzdial and Elliot Soloway. Teaching the Nintendo generation to program. *Commun. ACM*, 45(4):17–21, 2002.

[92] Kelsey Van Haaster and Dianne Hagan. Teaching and learning with BlueJ: an evaluation of a pedagogical tool. In *Journal of Issues in Informing Science and Information Technology*, pages 455–470, Santa ROsa, CA, 2004. IISIT.

[93] Brian Hanks and Matt Brandt. Successful and unsuccessful problem solving approaches of novice programmers. In *SIGCSE '09: Proceedings of the 40th ACM technical symposium on Computer science education*, pages 24–28, New York, NY, USA, 2009. ACM.

[94] Robert W. Hasker. An introductory programming environment for LEGO Mindstorms Robots, cited January 2009.

[95] Jesse M. Heines and Martin J. Schedlbauer. Teaching object-oriented concepts through GUI programming. In *ECOOP '07: Proceedings of the 11th Workshop on Pedagogies and Tools for the Teaching and Learning of Objects Oriented Concepts*, 2007.

[96] Poul Henriksen and Michael Kölling. Greenfoot: combining object visualisation with interaction. In *OOPSLA '04: Companion to the 19th annual ACM SIGPLAN conference on Object-oriented programming systems, languages, and applications*, pages 73–82, New York, NY, USA, 2004. ACM.

[97] Hi-tech outsourcing services. `http://www.hitechos.com/`.

[98] Elizabeth V. Howard, Donna Evans, Jill Courte, , and Cathy Bishop-Clark. A qualitative look at Alice and pair-programming. In *In The Proceedings of ISECON 2006*, volume 23, Dallas, TX, USA, 2006.

[99] Timothy Huang. Strategy game programming projects. In *CCSC '01: Proceedings of the sixth annual CCSC Northeastern Conference on The Journal of Computing in Small Colleges*, pages 205–213, , USA, 2001. Consortium for Computing Sciences in Colleges.

[100] Christopher D. Hundhausen, Saraha A. Douglas, and John T. Staskoz. A meta-study of algorithm visualization effectiveness. *Journal of Visual Languages and Computing*, 13:259–290, 2002.

[101] Letizia Jaccheri and Thomas Osterlie. Open source software: A source of possibilities for software engineering education and empirical software engineering. *First International Workshop on Emerging Trends in FLOSS Research and Development (FLOSS'07): ICSE Workshops 2007*, page 5, 2007.

[102] Andrea James. Computer science classes gets their groove back. Seattle Post-Intelligencer, 26 March 2009, 2009.

[103] Java Power Tools, cited July 2009. http://www.ccs.neu.edu/jpt/jpt_2_4/index.htm.

[104] Java Task Force Library, cited April 2009. http://jtf.acm.org/.

[105] Javabat: Java Practice, cited April 2009. http://www.javabat.com/.

[106] JCreator, cited April 2009. http://www.jcreator.com/.

[107] Jeliot 3, cited April 2009. http://cs.joensuu.fi/~jeliot/.

[108] Jeroo website, cited April 2009. http://home.cc.gatech.edu/dorn/jeroo.

[109] jGRASP, cited April 2009. http://www.jgrasp.org/index.html.

[110] Larry J. Stephens John Konvalina, Stanley A. Wileman. Math proficiency: a key to success for computer science students. *Communications of the ACM*, 26(5):377–382, 1998.

[111] Burke Johnson and Larry Christensen. *Educational Research: Quantitative, Qualitative, and Mixed Approaches, 2nd ed.* Pearson Education, Inc., 2004.

[112] Junit Testing Framework, cited April 2009. http://www.junit.org/home.

[113] Giuseppe Jurman, Stefano Merler, Annalisa Barla, Silvano Paoli, Antonio Galea, and Cesare Furlanello. Algebraic stability indicators for ranked lists in molecular profiling. *Bioinformatics*, 24(4):258–264, 2008.

[114] Colleen Kehoe, John Stasko, and Ashley Taylor. A survey of successful evaluations of program visualization and algorithm animation systems. *International Journal of Human-Computer Studies*, 54(2):265–284, 2001.

[115] Caitlin Kelleher. *Motivating Programming: Using storytelling to make computer programming attractive to middle school girls.* PhD thesis, Carnegie Mellon Universoty, 2006.

[116] Caitlin Kelleher and Randy Pausch. Lowering the barriers to programming: A taxonomy of programming environments and languages for novice programmers. *ACM Comput. Surv.*, 37(2):83–137, 2005.

[117] Caitlin Kelleher and Randy Pausch. Using storytelling to motivate programming. *Commun. ACM*, 50(7):58–64, 2007.

[118] Pivi Kinnunen and Lauri Malmi. Why students drop out CS1 course? In *Proceedings of the 2006 international workshop on Computing education research*, 2006.

[119] Changing Minds, Kolb's Learning Styles, cited July 2009. `http://changingminds.org/explanations/learning/kolb_learning.htm`.

[120] Alice Y. Kolb and David A. Kolb. The Kolb Learning Style Inventory-Version 3.1, 2005 Technical Specifications, 2005.

[121] Alexander Koller and Geert-Jan M. Kruijff. Talking robots with LEGO Mindstorms. In *COLING '04: Proceedings of the 20th international conference on Computational Linguistics*, page 336, Morristown, NJ, USA, 2004. Association for Computational Linguistics.

[122] Svetlana Kouznetsova. Using BlueJ and blackjack to teach object-oriented design concepts in CS1. *J. Comput. Small Coll.*, 22(4):49–55, 2007.

[123] Michael Külling and Poul Henriksen. Game programming in introductory courses with direct state manipulation. *SIGCSE Bull.*, 37(3):59–63, 2005.

[124] Michael Külling, Bruce Quig, Andrew Patterson, and John Rosenberg. The BlueJ system and its pedagogy. *Computer Science Education*, 13(4):249–268, 2003.

[125] Stan Kurkovsky. Engaging students through mobile game development. *SIGCSE Bull.*, 41(1):44–48, 2009.

[126] Essi Lahtinen, Hannu-Matti Järvinen, and Suvi Melakoski-Vistbacka. Targeting program visualizations. In *ITiCSE '07: Proceedings of the 12th annual SIGCSE*

conference on Innovation and technology in computer science education, pages 256–260, New York, NY, USA, 2007. ACM.

[127] Catherine Lang, Judy McKay, and Sue Lewis. Seven factors that influence ICT student achievement. *ACM SIGCSE Bulletin*, 39(3):221–225, 2007.

[128] Larry Latour. Microworlds, cited April 2009. `http://www.umcs.maine.edu/~larry/microworlds/microworld.html`.

[129] Matti Lattu, Jorma Tarhio, and Veijo Meisalo. How a visualization tool can be used - evaluating a tool in a research and development project. In *Proceedings of the 12th Workshop of the Psychology of Programming Interest Group*, 2000.

[130] Pamela B. Lawhead, Michaele E. Duncan, Constance G. Bland, Michael Goldweber, Madeleine Schep, David J. Barnes, and Ralph G. Hollingsworth. A road map for teaching introductory programming using LEGO©Mindstorms robots. In *ITiCSE-WGR '02: Working group reports from ITiCSE on Innovation and technology in computer science education*, pages 191–201, New York, NY, USA, 2002. ACM.

[131] Lucas Layman, Travis Cornwell, Laurie Williams, and Jason Osborne. Personality profiles and learning styles of advanced undergraduate computer science students. `ftp://ftp.ncsu.edu/pub/unity/lockers/ftp/csc_anon/tech/2005/TR-2005-40.pdf`.

[132] R. Mark Leeman and David H. Glass. Teaching Java with robots and artificial life. *Innovation in Teaching And Learning in Information and Computer Sciences*, 6(4):24–34, 2007.

[133] Scott Leutenegger and Jeffrey Edgington. A games first approach to teaching introductory programming. In *SIGCSE '07: Proceedings of the 38th SIGCSE technical symposium on Computer science education*, pages 115–118, New York, NY, USA, 2007. ACM.

[134] Eugene Levner, David Alcaide, and Joaquin Sicilia. Multi-attribute Text Classification Using the Fuzzy Borda Method and Semantic Grades. In *WILF '07: Proceedings of the 7th international workshop on Fuzzy Logic and Applications*, pages 422–429, Berlin, Heidelberg, 2007. Springer-Verlag.

[135] Ronit Ben-Bassat Levy and Mordechai Ben-Ari. We work so hard and they don't use it: Acceptance of software tools by teachers. In *ITiCSE '07: Proceedings of the 12th annual SIGCSE conference on Innovation and technology in computer science education*, pages 246–250, New York, NY, USA, 2007. ACM.

[136] Ronit Ben-Bassat Levy, Mordechai Ben-Ari, and Pekka A. Uronen. The Jeliot 2000 program animation system. *Comput. Educ.*, 40(1):1–15, 2003.

[137] Gary Lewandowski, Alicia Gutschow, Robert McCartney, Kate Sanders, and Dermot Shinners-Kennedy. What novice programmers don't know. In *ICER '05: Proceedings of the first international workshop on Computing education research*, pages 1–12, New York, NY, USA, 2005. ACM.

[138] Raymond Lister. Teaching Java first: experiments with a pigs-early pedagogy. In *ACE '04: Proceedings of the sixth conference on Australasian computing education*, pages 177–183, Darlinghurst, Australia, Australia, 2004. Australian Computer Society, Inc.

[139] Raymond Lister. Grand Challenges. *ACM SIGCSE Bulletin*, 37(2):14–15, 2005.

[140] Raymond Lister, Anders Berglund, Ilona Box, Chris Cope, Arnold Pears, Chris Avram, Mat Bower, Angela Carbone, Bill Davey, Michael de Raadt, Bernard Doyle, Sue Fitzgerald, Linda Mannila, Cat Kutay, Mia Peltomäki, Judy Sheard, Simon, Ken Sutton, Des Traynor, Jodi Tutty, and Anne Venables. Differing ways that computing academics understand teaching. In *ACE '07: Proceedings of the ninth Australasian conference on Computing education*, pages 97–106, Darlinghurst, Australia, Australia, 2007. Australian Computer Society, Inc.

[141] Raymond Lister, Anders Berglund, Tony Clear, Joe Bergin, Kathy Garvin-Doxas, Brian Hanks, Lew Hitchner, Andrew Luxton-Reilly, Kate Sanders, Carsten Schulte, and Jacqueline L. Whalley. Research perspectives on the objects-early. In *ITiCSE-WGR '06: Working group reports on ITiCSE on Innovation and technology in computer science education*, pages 146–165, New York, NY, USA, 2006. ACM.

[142] Yudong Liu. How algorithm animations aid learning in computer algorithm education - a literature review of algorithm animations in computer science education. `http://www.sfu.ca/~yudongl/_private/LiteratureReview-v3.pdf`.

[143] Torben Lorenzen and Ward Heilman. CS1 and CS2: Write computer games in Java! *SIGCSE Bull.*, 34(4):99–100, 2002.

[144] Linxiao Ma, John Ferguson, Marc Roper, and Murray Wood. Improving the viability of mental models held by novice programmers. In *ECOOP '07: Proceedings of the 11th Workshop on Pedagogies and Tools for the Teaching and Learning of Objects Oriented Concepts*, 2007.

[145] David J. Malan and Henry H. Leitner. Scratch for budding computer scientists. In *SIGCSE '07: Proceedings of the 38th SIGCSE technical symposium on Computer science education*, pages 223–227, New York, NY, USA, 2007. ACM.

[146] Mary Lynn Manns. *An Investigation into Factors Affecting the Adoption and Diffusion of Software Patterns in Industry*. PhD thesis, De Montfort University, 2002.

[147] Jane Margolis and Allan Fisher. *Unlocking the Clubhouse: Women in Computing*. MIT Press, Cambridge, MA, 2003.

[148] John Markoff. Computer science programs make a comeback in enrollment. NY Times, 16 March 2009, 2009.

[149] Michael McCracken, Vicki Almstrum, Danny Diaz, Mark Guzdial, Dianne Hagan, Yifat Ben-David Kolikant, Cary Laxer, Lynda Thomas, Ian Utting, and Tadeusz

Wilusz. A multi-national, multi-institutional study of assessment of programming skills of first-year CS students. In *ITiCSE-WGR '01: Working group reports from ITiCSE on Innovation and technology in computer science education*, pages 125–180, New York, NY, USA, 2001. ACM.

[150] Andrew McGettrick, Robert Boyle, Roland Ibbett, John Lloyd, Gillian Lovegrove, and Keith Mander. Grand Challenges in Computing: Education. Technical report, British Computer Society, Swindon, Wiltshire, SN1 1HJ, 2004.

[151] Jerry Mead, Simon Gray, John Hamer, Richard James, Juha Sorva, Caroline St. Clair, and Lynda Thomas. A cognitive approach to identifying measurable milestones for programming skill acquisition. *ITiCSE-WGR '06:Working group reports on ITiCSE on Innovation and technology in computer science education*, 2006.

[152] Mediacomp, cited April 2009. `http://coweb.cc.gatech.edu/mediaComp-plan/26`.

[153] Andrew Mertz, William Slough, and Nancy Van Cleave. The ACM Java libraries: Post-conference Workshop. *J. Comput. Small Coll.*, 24(1):127–128, 2008.

[154] Andrew Mertz, William Slough, and Nancy Van Cleave. Using the ACM Java libraries in CS 1. *J. Comput. Small Coll.*, 24(1):16–26, 2008.

[155] R. Mark Meyer and Debra T. Burhans. Robotran: A programming environment for novices using LEGO Mindstorms robots. In *Proceedings of the twenty-first AAAI Conference on Artificial Intelligence*. Association for Artificial Intelligence, 2006.

[156] Mirriam-webster's online dictionary, cited April 2009. `http://www.merriam-webster.com/dictionary/success`.

[157] Andrés Moreno, Niko Myller, Erkki Sutinen, and Mordechai Ben-Ari. Visualizing programs with Jeliot 3. In *AVI '04: Proceedings of the working conference on Advanced visual interfaces*, pages 373–376, New York, NY, USA, 2004. ACM.

[158] J. Morice. Skills and preferences: Learning from the Nintendo generation. In *International Workshop on Advanced Learning Technologies*, New York, NY, USA, 2000. IWALT.

[159] Briana B. Morrison and Jon A. Preston. Engagement: gaming throughout the curriculum. In *SIGCSE '09: Proceedings of the 40th ACM technical symposium on Computer science education*, pages 342–346, New York, NY, USA, 2009. ACM.

[160] Barbara Moskal, Deborah Lurie, and Stephen. Evaluating the effectiveness of a new instructional approach. In *SIGCSE '04: Proceedings of the 35th SIGCSE technical symposium on Computer science education*, pages 75–79, New York, NY, USA, 2004. ACM.

[161] Paul Mullins, Deborah Whitfield, and Michael Conlon. Using Alice 2.0 as a first language. *J. Comput. Small Coll.*, 24(3):136–143, 2009.

[162] Thomas L. Naps, Guido Rößling, Vicki Almstrum, Wanda Dann, Rudolf Fleischer, Chris Hundhausen, Ari Korhonen, Lauri Malmi, Myles McNally, Susan Rodger, and J. Ángel Velázquez-Iturbide. Exploring the role of visualization and engagement in computer science education. In *ITiCSE-WGR '02: Working group reports from ITiCSE on Innovation and technology in computer science education*, pages 131–152, New York, NY, USA, 2002. ACM.

[163] NetBeans, cited April 2009. `http://www.netbeans.org/`.

[164] Jill Van Newenhizen. The Borda Method is most likely to respect the Condorcet Principle. *Economic Theory*, 2(1):69–83, 1992.

[165] Lijun Ni. What makes CS teachers change?: Factors influencing CS teachers' adoption of curriculum innovations. In *SIGCSE '09: Proceedings of the 40th ACM technical symposium on Computer science education*, pages 544–548, New York, NY, USA, 2009. ACM.

[166] ObjectDraw Library, cited April 2009. `http://eventfuljava.cs.williams.edu/library/`.

[167] Michael Olan. Dr. J vs. the bird: Java IDE's one-on-one. *J. Comput. Small Coll.*, 19(5):44–52, 2004.

[168] M. Frank Pajares. Teachers' beliefs and educational research: Cleaning up a messy construct. *Review of Educational Research*, 62(3):307–332, 1992.

[169] Deniz Palak and Richard T. Walls. Teachers' beliefs and technology practices: A mixed-methods approach. *Journal of Research on Technology in Education*, 41(4):417–441, 2009.

[170] Brenda Parker and Ian Mitchell. Effective methods for learning: a study in visualization. *J. Comput. Small Coll.*, 22(2):176–182, 2006.

[171] James H. Paterson, John Haddow, and Michael Nairn. A design patterns extension for the BlueJ IDE. In *ITICSE '06: Proceedings of the 11th annual SIGCSE conference on Innovation and technology in computer science education*, pages 280–284, New York, NY, USA, 2006. ACM.

[172] Andrew Patterson, Michael Kölling, and John Rosenberg. Introducing unit testing with BlueJ. In *ITiCSE '03: Proceedings of the 8th annual conference on Innovation and technology in computer science education*, pages 11–15, New York, NY, USA, 2003. ACM.

[173] Michael Quinn Patton. *Qualitative Evaluation and Research Methods*. Sage publications, Newbury Park, CA, 2nd edition, 1990.

[174] Arnold Pears, Stephen Seidman, Crystal Eney, Päivi Kinnunen, and Lauri Malmi. Constructing a core literature for computing education research. *SIGCSE Bull.*, 37(4):152–161, 2005.

[175] Arnold Pears, Stephen Seidman, Lauri Malmi, Linda Mannila, Elizabeth Adams, Jens Bennedsen, Marie Devlin, and James Paterson. A survey of literature on the teaching of introductory programming. *SIGCSE Bull.*, 39(4):204–223, 2007.

[176] Maureen L. Pope and Eileen M. Scott. *Teacher Thinking Twenty Years on: Revisiting Persisting Problems and Advances in Education*, chapter Teachers' epistemology and practice. Lisse:Swets and Zeitlinger, 2002.

[177] Kris Powers, Stacey Ecott, and Leanne M. Hirshfield. Through the looking glass: teaching CS0 with Alice. In *SIGCSE '07: Proceedings of the 38th SIGCSE technical symposium on Computer science education*, pages 213–217, New York, NY, USA, 2007. ACM.

[178] Kris Powers, Paul Gross, Steve Cooper, Myles McNally, Kenneth J. Goldman, Viera Proulx, and Martin Carlisle. Tools for teaching introductory programming: what works? In *SIGCSE '06: Proceedings of the 37th SIGCSE technical symposium on Computer science education*, pages 560–561, New York, NY, USA, 2006. ACM.

[179] Colin B. Price. From Kandinsky to Java (the use of 20th century abstract art in learning programming). *Innovation in Teaching And Learning in Information and Computer Sciences*, 6(4):35–50, 2007.

[180] Colin B. Price, John Colvin, and Warren Wright. Introducing game development into the computing curriculum - a progressive methodology. *Innovation in Teaching And Learning in Information and Computer Sciences*, 5(3), 2006.

[181] Jeff Raab, Richard Rasala, and Viera K. Proulx. Pedagogical power tools for teaching Java. In *ITiCSE '00: Proceedings of the 5th annual SIGCSE/SIGCUE ITiCSEconference on Innovation and technology in computer science education*, pages 156–159, New York, NY, USA, 2000. ACM.

[182] Noa Ragonis and Mordechai Ben-Ari. On understanding the statics and dynamics of object-oriented programs. In *SIGCSE '05: Proceedings of the 36th SIGCSE*

technical symposium on Computer science education, pages 226–230, New York, NY, USA, 2005. ACM.

[183] Shri Rai, Kok Wai Wong, and Peter Cole. Game construction as a learning tool. In *CyberGames '06: Proceedings of the 2006 international conference on Game research and development*, pages 231–236, Murdoch University, Australia, Australia, 2006. Murdoch University.

[184] Teemu Rajala, Mikko-Jussi Laakso, Erkki Kaila, and Tapio Salakoski. Effectiveness of program visualization: A case study with the ViLLE tool. *Journal of Information Technology Education: Innovations in Practice*, 7:15–32, 2008.

[185] Yolanda Rankin, Amy Gooch, and Bruce Gooch. The impact of game design on students' interest in CS. In *GDCSE '08: Proceedings of the 3rd international conference on Game development in computer science education*, pages 31–35, New York, NY, USA, 2008. ACM.

[186] Richard Rasala, Jeff Raab, and Viera K. Proulx. Java Power Tools: Model software for teaching object-oriented design. In *SIGCSE '01: Proceedings of the thirty-second SIGCSE technical symposium on Computer Science Education*, pages 297–301, New York, NY, USA, 2001. ACM.

[187] Stuart Reges. The mystery of b := (b = false). *ACM SIGCSE Bulletin*, 40(1):21–25, 2008.

[188] Charles Reis and Robert Cartwright. A friendly face for Eclipse. In *Eclipse '03: Proceedings of the 2003 OOPSLA workshop on eclipse technology exchange*, pages 25–29, New York, NY, USA, 2003. ACM.

[189] Charles Reis and Robert Cartwright. Taming a professional IDE for the classroom. In *SIGCSE '04: Proceedings of the 35th SIGCSE technical symposium on Computer science education*, pages 156–160, New York, NY, USA, 2004. ACM.

[190] Renovating schools, cited April 2009. `http://www.nap.edu/html/techgap/nintendo.html`.

[191] Mitchel Resnick and Brian Silverman. Some reflections on designing construction kits for kids. In *IDC '05: Proceedings of the 2005 conference on Interaction design and children*, pages 117–122, New York, NY, USA, 2005. ACM.

[192] Lauren Rich, Heather Perry, and Mark Guzdial. A CS1 course designed to address interests of women. In *SIGCSE '04: Proceedings of the 35th SIGCSE technical symposium on Computer science education*, pages 190–194, New York, NY, USA, 2004. ACM.

[193] Robotran, cited April 2009. `http://www.prm.ucl.ac.be/robotran/indexEN.html`.

[194] Ma. Mercedes T. Rodrigo, Ryan S. Baker, Matthew C. Jadud, Anna Christine M. Amarra, Thomas Dy, Maria Beatriz V. Espejo-Lahoz, Sheryl Ann L. Lim, Sheila A.M.S. Pascua, Jessica O. Sugay, and Emily S. Tabanao. Affective and behavioral predictors of novice programmer achievement. In *ITiCSE '09: Proceedings of the 14th annual ACM SIGCSE conference on Innovation and technology in computer science education*, pages 156–160, New York, NY, USA, 2009. ACM.

[195] E.M. Rogers. *Diffusion of Innovations, 4th Edition*. The Free Press, 1995.

[196] John Rosenberg and Michael Kölling. Testing object-oriented programs: Making it simple. In *SIGCSE '97: Proceedings of the twenty-eigth SIGCSE technical symposium on Computer science education*, pages 77–81, New York, NY, USA, 1996. ACM.

[197] Nathan Rountree, Janet Rountree, and Anthony Robins. Predictors of success and failure in a CS1 course. *ACM SIGCSE Bulletin*, 38(1):121–124, 2002.

[198] Nathan Rountree, Janet Rountree, Anthony Robins, and Robert Hannah. Interacting factors that predict success and failure in a CS1 course. In *ITiCSE-WGR*

'04: Working group reports from ITiCSE on Innovation and technology in computer science education, pages 101–104, New York, NY, USA, 2004. ACM.

[199] Dean Sanders and Brian Dorn. Classroom experience with Jeroo. *J. Comput. Small Coll.*, 18(4):308–316, 2003.

[200] Dean Sanders and Brian Dorn. Jeroo: a tool for introducing object-oriented programming. In *SIGCSE '03: Proceedings of the 34th SIGCSE technical symposium on Computer science education*, pages 201–204, New York, NY, USA, 2003. ACM.

[201] Purvi Saraiya, Clifford A. Shaffer, D. Scott McCrickard, and Chris North. Effective features of algorithm visualizations. *SIGCSE Bull.*, 36(1):382–386, 2004.

[202] Dino Schweitzer and Wayne Brown. Interactive visualization for the active learning classroom. In *SIGCSE '07: Proceedings of the 38th SIGCSE technical symposium on Computer science education*, pages 208–212, New York, NY, USA, 2007. ACM.

[203] Vijayakumar Shanmugasundaram, Paul Juell, Curt Hill, and Kendall Nygard. Effectiveness of BlueJ in learning Java. In Craig Montgomerie and Jane Seale, editors, *Proceedings of World Conference on Educational Multimedia, Hypermedia and Telecommunications 2007*, pages 3776–3781, Vancouver, Canada, June 2007. AACE.

[204] Robert H. Sloan and Patrick Troy. CS 0.5: a better approach to introductory computer science for majors. In *SIGCSE '08: Proceedings of the 39th SIGCSE technical symposium on Computer science education*, pages 271–275, New York, NY, USA, 2008. ACM.

[205] Elliot Soloway. How the Nintendo generation learns. *Commun. ACM*, 34(9):23–ff., 1991.

[206] Robert E. Stake. Progressive focusing. cited April 2009, cited April 2009.

[207] Christine Stephensen. *Educational Technology Associations as Change Agents: A Case Study.* PhD thesis, Oregon State University, 2007.

[208] Zweben Stuart. PhD. production exceeds 1,700; undergraduate enrollment trends still unclear. *Computing Research News*, 20(3), 2008.

[209] Jay Summet, Deepak Kumar, Keith O'Hara, Daniel Walker, Lijun Ni, Doug Blank, and Tucker Balch. Personalizing CS1 with robots. In *SIGCSE '09: Proceedings of the 40th ACM technical symposium on Computer science education*, pages 433–437, New York, NY, USA, 2009. ACM.

[210] SurveyMonkey, cited February 2009. `http://www.msurveymonkey.com`.

[211] Elizabeth Sweedyk and Robert M. Keller. Fun and games: a new software engineering course. In *ITiCSE '05: Proceedings of the 10th annual SIGCSE conference on Innovation and technology in computer science education*, pages 138–142, New York, NY, USA, 2005. ACM.

[212] Charles Teddie and Abbas Tashakkori. *Foundations of Mixed Methods Research: Integrating quantitative and qualitative approaches in the social and behavior sciences*. Sage Publications, Inc., Thousand Oaks, CA, USA, 2009.

[213] Allison Elliot Tew, W. Michael McCracken, and Mark Guzdial. Impact of alternative introductory courses on programming concept undersanding. In *ICER 05: Proceedings of the 2005 international workshop on Computing education research*, 2005.

[214] Lynda Thomas, Mark Ratcliffe, John Woodbury, and Emma Jarman. Learning styles and performance in the introductory programming sequence. *SIGCSE Bull.*, 34(1):33–37, 2002.

[215] Sheila Tobias. *They're Not Dumb, They're Different, Stalking the Second Tier*. Research Corporation, a foundation for the advancement of science, 1990.

[216] Markku Tukiainen and Eero Monkkonen. Programming aptitude testing as a prediction of learning to program. In *Proceedings of the 14th Workshop of the Psychology of Programming Interest Group*, 2002.

[217] Jaroslav Tulach, Rich Unger, and Timothy Boudreau. Decoupled design: building applications on the NetBeans platform. *Companion to the 21st ACM SIGPLAN conference on Object-oriented programming languages*, page 854, 2006.

[218] Jaime Urquiza-Fuentes and J. Ángel Velázquez-Iturbide. A survey of successful evaluations of program visualization and algorithm animation systems. *Trans. Comput. Educ.*, 9(2):1–21, 2009.

[219] Jay Vegso. Enrollments and degree production at US CS departments drop further in 2006-07. *Computing Research News*, 20(2), 2008.

[220] Tamar Vilner, Ela Zur, and Judith Gal-Ezer. Fundamental concepts of CS1: procedural vs. object oriented paradigm - a case study. In *ITiCSE '07: Proceedings of the 12th annual SIGCSE conference on Innovation and technology in computer science education*, pages 171–175, New York, NY, USA, 2007. ACM.

[221] Fang Wei, Sally H. Moritz, Shahida M. Parvez, and Glenn D. Blank. A student model for object-oriented design and programming. *J. Comput. Small Coll.*, 20(5):260–273, 2005.

[222] Sandra Wescott. Effectiveness of using digital game playing in a first-level programming course, 2008. D.P.S. Dissertation, Pace University.

[223] Keith J. Whittington. Infusing active learning into introductory programming courses. *J. Comput. Small Coll.*, 19(5):249–259, 2004.

[224] Richard Wicentowski and Tia Newhall. Using image processing projects to teach CS1 topics. In *SIGCSE '05: Proceedings of the 36th SIGCSE technical symposium on Computer science education*, pages 287–291, New York, NY, USA, 2005. ACM.

[225] Susan Wiedenbeck. Factors affecting the success of non-majors in learning to program. In *ICER 05: Proceedings of the 2005 international workshop on Computing education research*, 2005.

[226] Brenda Cantwell Wilson. A study of factors promoting success in computer science including gender differences. *Computer Science Education*, 12(1):141 – 164, 2004.

[227] Brenda Cantwell Wilson and Sharon Schrock. Contributing to success in an introductory computer science course: A study of twelve factors. *ACM SIGCSE Bulletin*, 33(1):184–188, 2001.

[228] Shengli Wu and Fabio Crestani. Ranking retrieval systems with partial relevance judgements. *Journal of Universal Computer Science*, 14(7):1020–1030, 2008.

[229] Stelios Xinogalos, Maya Satratzemi, and Vassilios Dagdilelis. Teaching Java with BlueJ: a two-year experience. In *ITiCSE '07: Proceedings of the 12th annual SIGCSE conference on Innovation and technology in computer science education*, pages 345–345, New York, NY, USA, 2007. ACM.

[230] Stelios Xinogalosa, Maya Satratzemia, and Vassilios Dagdilelisb. An introduction to object-oriented programming with a didactic microworld: ObjectKarel. *Computers & Education*, 47(2):148–171, 2004.

[231] Allan Yuen. Teaching computer programming: A connectionist view of pedagogical change. *Australian Journal of Education*, 2000.

[232] Imran A. Zualkernan. Using Soloman-Felder Learning Style Index to Evaluate Pedagogical Resources for Introductory Programming Classes. In *ICSE '07: Proceedings of the 29th international conference on Software Engineering*, pages 723–726, Washington, DC, USA, 2007. IEEE Computer Society.

[233] James E. Zull. *The Art of Changing the Brain*. Stylus Publishing, Sterling, VA, 2002.