https://doi.org/10.48009/2_iis_2009_128-137

Selecting a first programming Language to Teach Prospective Teachers – Case Examples from Two Programs

Azad Ali, Indiana University of Pennsylvania, azad.ali@iup.edu
Frederick Kohun, Robert Morris University, Kohun@rmu.edu
David Wood, Robert Morris University, Wood@rmu.edu

ABSTRACT

The purpose of this paper is to identify the factors that influence the decision to select a programming language to teach students enrolled in a graduate level course in education. The paper illustrates the experience of two degree programs: First, the master degree of business education (M.Ed.) at Eberly College of Business and Information Technology – Indiana University of Pennsylvania (IUP), and second, the master of education (M.S. Business Education) at Robert Morris University (RMU). The paper first reviews literature regarding factors that make learning to program a difficult task. It then introduces a programming language named Alice and discusses how this language is able to address the difficulty with learning to program. It then illustrates the experience of both programs at IUP and RMU in selecting a programming language to teach for their students enrolled in their respective education master degree programs.

Keywords: Introduction to Programming, Programming for teachers, Programming for high schools, Programming for prospective teachers, Beginner programming language

INTRODUCTION

Academic technology programs often struggle with the selection of a programming language for entry level programming courses. An entry level programming course can serve as a launching pad for other more advanced programming courses. It sets the stage to learn more advanced programming topics. At the same time, a beginning programming course can also be taken by students from other non-technology majors. While the non-major students may not have the same interest in preparing to learn advanced programming topics, it must be taken into consideration that many students may have dissimilar backgrounds with respect to students enrolled in technology programs.

One of the first steps that are usually taken when deciding on the contents for a beginner programming course is the selection of a programming language used to teach the course. Programming languages vary from one program to another. Their purpose of introduction to courses may be different also. Teachers may choose from beginner programming languages (such as Alice and Scratch) to fit the entry level student need. The same teachers may also select from general purpose programming languages (such as Java, C++ and Visual Basic) to fit a wider range of applications. Consideration for a diverse population of students enrolled in the course as well as student perception of course difficulty must be included in the decision process for selecting the programming language for an entry level programming course.

This paper illustrates the experience of two programs in selecting a programming language to teach master degree students in a first programming course. The M.Ed. program at IUP and the M.S. Business Education program at RMU both teach programming topic/course for their students enrolled in their respective education master degree programs. The paper begins by explaining the factors that make learning to program a difficult task. It then introduces a programming language named Alice and shows how this programming language addresses identified difficulty factors. It then introduces other factors that usually influence the decision to selecting the programming language. The paper then elaborates on the experience of the two programs at IUP and RMU and how they addressed these factors when selecting a programming language for their students enrolled in their graduate programs. A summary of the paper is presented at the end.

Difficulty Reasons for Learning to Program

Learning to program is considered to be a difficult task to many students. It is estimated that at least 25% to 80% of students drop their first computer course due to the difficulty in learning to program [4]. This difficulty is experienced by students enrolled in technology who are required to take programming courses as well as other students who take programming courses as part of their requirements to complete their degree [3].

Other studies have analyzed the factors that contribute to the difficulty of learning to program and suggested remedies to these difficulties. Dann, Cooper and Pausch [4] for example listed four factors that contribute to the difficulty associated with learning and programming: Fragile mechanics of program creation, particularly syntax; the inability to see the result of computation as the program runs, the lack of motivation for programming and the difficulty of understanding compound logic and learning design techniques.

In a study conducted to suggest steps to simplify learning to program, Kelleher and Pausch [8] compared a number of programming languages that are commonly used in beginner programming courses. The same study wrote the following about the difficulty of learning to program:

> Learning to program can be very difficult for beginners of all ages. In addition to the challenges of learning to form structured solutions to problems and understanding how programs are executed, beginning programmers also have to learn a rigid syntax and rigid commands that may have seemingly and arbitrary or perhaps confusing names. Tackling all of these challenges simultaneously can be overwhelming and often discouraging for beginning programmers (p. 83).

The studies listed above point to a common fact: learning to program for a beginner is considered to be a difficult task. However, the factors that contribute to these difficulties are not totally agreed upon. Thus, further explanations of these factors may shed some lights on the specific reasons for the difficulty. The remainder of this section explains in more detail some of the reasons that contribute to the difficulty in learning to program.

Fragile Mechanics and Syntax

The mechanics of developing a program are not standardized. It is called fragile because there is no direct and pre-defined way to develop a program. Although program creation usually starts by learning the syntax, some argue that tackling the syntax is not the best way to learn to program. Learning syntax is foreign to some students and, as a result, they may spend a lot of time learning the programming language syntax without context.

Syntax is “the grammatical role of the programming language” or so explained in typical programming courses. However, a closer look at the rules of syntax in programming languages reveals many differences from the grammatical rules of typical natural language. These differences have to do with the structure of commands, the stopping character, naming of variables, passing of parameters and other related issues when structuring lines in programs.

The error messages that are generated by the compiler are not always helpful. Sometime, these error messages are designed for advanced programmers and the wording of the messages do not help beginning programmers understand their meaning. Other times, the error messages may point to a particular line of code while the actual error is at a previous or a totally different location within the program. In these cases, the beginning programmers keep looking at the line where the error message appears and can’t identify the actual error. Locating and correcting syntax errors may then, as a result, end up being a long and daunting task. For beginners who are not familiar with programming, spending such a long time on fixing simple syntactical errors can become frustrating

Compound Logic and Structure

Teaching program design and structure is not new in academia. Structure has been used in different fields of study and provides advantages when used in a particular field. In computer programming, the term “structure” is repeated often and is practiced differently when writing programs. Actually, the word structure is considered the foundation in three different flows of code when writing different programs. These three different procedures for controlling the flow of code are termed the three control structures: sequence control structure, selection control structure and iteration control structure. The level of "structure" is practiced differently in each of the three "control structures" and, as a result, the associated difficulty in understanding them may vary also.

The reference to structure in programming is more than just the three control structures: it references the design of a program that breaks down the logic into meaningful and manageable modules. The difficulty and complexity arises as the number of modules and the links and connections between the different modules increases.

This call for structure is designed to make a program development cycle more efficient and is aimed at standardizing the coding of programs and reusing existing code of the programs. However, this kind of structure is unfamiliar to inexperienced programmers. Later developments in the programming industry introduced the use of Object Oriented Programming (OOP) methodology which stresses revising the issue of structure in program development. The OOP methodology introduced many new concepts that needed to be understood along with its associated programming concepts. Dann, Copper and Pausch [4] noted that today's beginning programmers have to learn the original concepts of programming such as loop, selection, and iteration along with the new concepts of OOP such as classes, objects, encapsulation, inheritance, and others. Thus, it adds additional steps to learn which, as a result, adds to the difficulty of learning to program.

Seeing Program Results

In order to see the result of execution from any program, the program more often needs to be error free, to execute, and then, display the output. However, making any program run is not always a simple process. After working through the syntax and making the program start to run, the programmer then faces additional errors that are called "execution errors". These errors may result from missing variable name or using different field types (numeric or otherwise) for the wrong purpose (such as using non-numeric fields for computations or dividing by zero). After working through all these steps, the programmer still may not be able to see the final program output.

The steps listed above did not include working through what is termed "logical errors". These are the errors that result from incorrect program logic. These errors are often harder to locate and correct and may require a deeper understanding of the application problem as well as more programming knowledge before being able to correct them. In this case, similar to syntax error messages, the logical and execution error messages that are generated from the compiler are not very helpful to beginning programmer. As a result, programmers have to sift through their lines of code in order to find what caused the execution or logical error.

The main difficulty with these sequence of steps is that most often the programmer is unable to see the result of the program until going through all the errors (syntax, execution and logical errors). In some cases, there are no visual clues or intermediate steps to show a preliminary output. Often, the programmer has to repeat many steps that take a long time before seeing a result of the program.

Lack of Motivation

The programming profession is seen by many people as a boring job (Rosamita, 2007). This point of view stems from the notion that programmers sit in front of a computer spending long hours trying to produce output or correct errors that seem to be of minimal importance. Thus, motivation is minimal to take different programming courses that have the potential to become qualified to take on what it seems to be a boring job.

A common first program that is used during an entry level programming courses displays a message that prints "Hello World" to the audience. Additional typical "first" programming assignment examples may include writing a program to convert Fahrenheit to Centigrade or converting miles driven to kilometers.

Writing the programs to produce the examples mentioned above may follow different steps when using one programming language versus another. However, it is safe to say that producing small outputs like the ones described above take a number of steps and a certain amount of time. To beginning students who are taking a first programming course, putting this kind of effort to produce a simple output may not be justified and may not be time efficient.

Students may question the feasibility of spending this much time to produce simple outputs that are generated from writing programs. Added to all of that, the process of finding and correcting logical errors requires a certain level of understanding of the logic and structure of the program. As a result, students often may be less motivated to get enrolled in programming courses that potentially get them in what it seems to be "boring" jobs.

Alice Programming Language

This section explains about the use of a new language in introductory programming course. It begins by providing a brief explanation of Alice programming language and details how this language addresses difficulty factors for learning to program that were identified earlier in this paper. The paper then explains the factors that make Alice as a good candidate for the language to teach prospective teachers in introductory programming courses.

Alice is a programming language that was introduced by Carnegie Mellon University and it seems that it has provided the answers to the questions that were raised about the difficulty of programming languages. Alice provides a visual interface that makes it easier to follow, and it cuts down on the syntax and coding.

Alice has increased in popularity for use in first year programming courses at both colleges and high schools. The increasing popularity of Alice as a first programming language is due to the many advantages that it provides over traditional or general purpose programming languages. Adams [1] noted thee advantages to using Alice in introductory programming courses:

> The allure of 3D graphics. It is difficult to overstate the visual appeal of 3D animations, especially to today's visually-oriented students. When you program works, you feel euphoric! But even when you make a mistake (a logic error), the results are often comical, producing laughter instead of frustration.
>
> The Alice IDE. Alice includes a drag-and—drop integrated development environment (IDE) that eliminates syntax errors. The IDE eliminates all of the missing semicolons, curly braces, quotation marks, misspelled keywords or identifiers, and other syntax problems that bedevil CS1 students.

The design of Alice aims at addressing the points of difficulty that is often faced by students taking a first programming course as explained in the remainder of this section.

Alice Syntax and Mechanism

When developing a program in Alice, users do not have to type the program. Instead, users pull down objects and align them according with specified commands that are already drawn for the user [11]. As the user pulls a particular object, another dropdown menu appears that gives the user options to choose from. The key here is that there is no room to make syntax errors when using Alice. Instead, efforts can be directed to understand the mechanism and the concepts of the program [10].

The mechanism of program development is more direct in Alice than in other programming languages. Program development in Alice begins by creating what is termed as "world" which is the stage for placing objects that work together to produce an intended output. All the objects are placed in a defined and visually apparent library, thus making it is easier to look them over and make them work. So the point that can be made here is: there are no syntax errors in Alice. As a result, students do not get frustrated looking over and over their code to find syntax errors. The other point that can be made here is that due to the "visual" nature of Alice program, the mechanism of developing a program is more visible and direct that other languages.

The Structure in Alice

Alice uses the structure of object oriented programming [9]. In Alice, classes are selected to select from and then objects from the classes are pulled to a "world". As each object is drawn on the world, a visible list of properties and methods can be observed and pulled so to use them in the program.

Classes are pulled from a class library. This class library contains a large collection of visual objects that are easily recognized and noticed. Once an object from a class is drawn in the editor area, the objects can be seen as having properties, methods and functions. These terms can be easier understood because they refer to characteristics of visual objects such as height, width, moving in one direction, distance and other similar characteristics.

Similar to other programming languages, Alice uses functions, methods, events and others. It passes parameters, receives output, and creates a structure to the program. All of this is done very similarly to most other programming languages except that Alice uses visual objects which are easier seen and understood [7].

As methods and functions are created in Alice, visual buttons are added to the program that easily identifies them. Reusability of the pre-written modules is a simple drag and drop from one location to another. The content of the module (whether properties, methods or others) are pulled with the module when dropping it to a different location. In other words, Alice enforces the use of OOP through visual, easily identifiable structures that have distinct characteristics and could be used along different programs to enforce the re-usability concept.

Seeing Programming Results

Programming in Alice enables individuals to see the programming code right after the program ends. As the objects are pulled off the visual library, the programmer is able to see results immediately. There are no syntax errors in Alice, thus programmers do not have to stumble through lines of syntax errors. As a result, programmers using Alice are able to see program result as soon as they finish or complete the program.

The issue of execution and logical errors are handled more easily in Alice than in other programming languages. The design of the program reduces the possibility of execution error. By using visual objects that can be easily identified along with their visual properties and methods, interim results of the program can be easily observed. Hence, some of the execution and logical errors can be identified and corrected during design time rather than wait until program completion when the program is longer and locating the error becomes less obvious.

The Issue of Motivation

Lack of motivation in learning to program comes from the notion that learning to program is boring and the time it takes to develop a simple output is considerably long. In other words, the time/output ratio is high enough to cause some to think that learning to program is not worth the time invested.

Alice uses visual output. All objects within Alice are three dimensional visual objects. The output that is usually generated from a typical Alice program, as a result, is more visually appealing. The objects represent popular metaphors which tell stories, draw shapes, and have moving components. These movements on the screen provide an interesting application to the programmer.

Development time for Alice programs is minimal compared to general purpose programming languages. Additionally, Alice engages the programmer during development times as well as during the testing phase. By using metaphors that are popular in society, the program will not be limited to displaying simple text output. Instead, the program generates objects that are jumping, talking, and changing color or similar techniques that are used in game development. In other words, working with Alice helps fade the notion that programming is “boring” while also it enhances motivation. The students behavior changes because they are able to create programs in less time that produce more interesting output.

Alice and Prospective Teachers

From the earlier discussion, it seems obvious that Alice is a programming language that fits the environment for teaching entry level programming courses. Alice also can be considered a language to be used for students enrolled in education programs for prospective teachers.

Prospective teachers are more likely to apply the knowledge they have gained from their formal education into their own classrooms. As so, students who graduate from education programs (or business education) develop a prospective on what they teach based on what they learned in their prior education. Therefore, it can be argued, Alice is an ideal language to teach prospective teachers because if they can understand it and apply it, they can carry it with them and teach it to their future students.

Using Alice in a first programming course may be able to solve many of the problems associated with teaching programming for high school students for the following reasons:

- Alice dispels the common notion that pervades high schools: learning to program is considered to be asocial [12]. Such a notion discourages most students in general (high school students in particular) from getting enrolled in computer programs.
- Using Alice, programming teachers can incorporate toys, games and other activities [1] that high school students want to have in their class more than other students.
- The Alice application is more likely to have characteristics that interest high school students more often than others; it uses characters that most high school students relate to [6].

Other Institutional Considerations

Selecting a programming language that simplifies the learning of students could help attracting more students into these courses. However, this is not the only factor that decides on which programming language to select in beginner programming courses. Instead, different other factors influence the selection of such programming language in entry level courses. This section discusses other considerations that may need to be taken into account when contemplating choosing a programming language for entry level programming courses.

General Purpose versus Beginner Programming Courses

One of the main issues that educators in introductory programming courses deal with is whether to select a programming language from the general purpose programming group or from the beginner level programming group. General purpose programming courses are those that have been used in the industry for a while such as Java, C++, Visual Basic and others. Beginner programming languages are the languages that are designed to simplify the learning of programming. They usually include tools and objects that make learning to program simpler. An example of this is the Alice programming language and Scratch.

Selecting a general programming course opens the door to wider range of applications and also opens the link to simplify learning other similar general purpose programming languages. However, a beginner programming language has a distinct advantage in that is it is designed for beginners, thus it simplifies learning requirements for students.

Major Only versus Service Courses

A beginner programming course may be taken by students with technology majors only, often designated a “major only” course. However, at the same time, the same course may be required for students from other departments and be referred to as a “service” course.

In the first case, having all the students in the course as technology majors provides the opportunity to teach advanced topics. In this case, general purpose programming languages are appropriate because they are known to be able to interface with other applications such as database, spreadsheets and operating system utilities—thereby opening the door to teach advanced topics.

However, if the course is a service course, the teacher may take into consideration ways to simplify the course. Non-technology major students may not have the same interest in learning advanced topics and interfacing with other applications as students from technology majors. In the case of a programming language as a service course, considerations for simplifying the learning of programming may take precedence, and therefore requiring the need for the selection of one of the beginner programming languages.

Entire Course versus Selected Topics

Programming courses often dedicate an entire course to teach one specific programming language. Other courses teach programming as a selected topic among different other topics within that course. Each of these cases affects the selection of programming language. If programming is taught as one of other topics in the course, less depth is allocated for the topic and, as a result, a beginner programming language may need to be selected. On the other hand, dedicating an entire course (or more than one course) to teach one programming language, advanced topics including the interaction with other applications (such as databases) may need to be addressed. Thus the selection of general purpose programming could provide wider area of coverage and utility in this case.

Standalone versus Prerequisites

If the course is a prerequisite to other courses, then faculty may have to select a language that insures adequate transition to the advanced course. For example, a beginner level programming course may teach Visual Basic. This course may be followed by another course such as "Advanced Programming in Visual Basic." The faculty has no choice but to teach this programming language to ensure smooth transition to the second and more advanced course.

In some cases, programs dedicate one programming course that serve as a prerequisite for other courses to teach other programming languages. For example, a program may have one course titled "Intro to Programming" at the 100 level and then "Programming in C++" at the 200 level with Intro to Programming as a course prerequisite. In this case and similar ones, it may be beneficial to teach a beginner programming language at the 100 level (or as the Intro course), to provide solid understanding of the concepts prior to taking as more advanced general purpose programming courses.

The Two Graduate Programs

This section explains about the two courses that teach the programming course/topic for their master degree students in education at Indiana University of Pennsylvania (IUP) and Robert Morris University (RMU). The section first discusses the degree program at IUP and then the degree program at RMU. It explains about each and addresses the issues associated with selecting a programming language to teach at their respective programs.

The M.Ed. Program at IUP

The Technology Support and Training (TST) department at Eberly College of Business in Indiana University of Pennsylvania (IUP) offers a master degree program in business education (M.Ed.). The main goal of this master degree is to prepare students to be teachers in the business and technology field. One particular course that is included in this master degree program is a capstone course called "BTST680 Technical Update". The course teaches the latest in technology and includes four categories or sub-coverage areas: Programming, Database, Digital Media and Networking. The following describes the selection of a programming language for this course and the methods in which it is being taught.

One of the main difficulties in teaching this course is that most of the students enrolled do not have prior programming experience. Some students may have had exposure to programming languages prior to this course, but the information is often outdated and forgotten. Due to these limitations, the course has to begin by teaching the principles of programming and the syntax and logic of programming.

Alice programming language was selected for this course to teach the programming topic. The main reason for selecting Alice is that the students in this course are prospective teachers. Therefore it will be useful to teach them this language as they may need it for their professional lives. The students in this course are not looking for a programming job in the industry; hence it will not help them to teach a general programming language such as Java or C++. Instead, they can focus on learning the concepts of programming by using the tools available in Alice.

The faculty member teaching this course has been using Alice for the past two years. The feedback from selecting Alice in this course has been positive and enrollment has increased in this course since introducing Alice it. The learning curve in the course has also increased. The students master the programming concepts quicker as compared to previous semesters. Students are required to complete and present a final comprehensive project with Alice. All presentations have been successful while the students showed good understanding of the programming terminology such as objects, properties, methods, encapsulation, and inheritance.

The M.S. Program in Business Education Program at RMU

Robert Morris University (RMU) offers a master degree program in Business Education. This program requires their students to take one programming language. The programming course is taught by the college of Information Systems and Communication. The title of the course is Visual Basic Programming and the course number is INFS6120.

The Visual Basic programming course is required by students who are enrolled in the technology programs at RMU. Though this course is not a formal prerequisite to other courses in the program, but there is a common understanding that this course is the first programming course for students enrolled in the M.S. Internet Information Systems program, M.S. Information Systems Management program ,and other technology programs. Other technology programs at the university require their students to take additional programming languages such as Java and C++. Though these upper level courses do not have explicit requirements to take the Visual Basic course, there is, however, implicit understanding among faculty and students that the Visual Basic course is a first programming course in the program.
Reactions from students taking this course have been mixed. Technology majors most often have positive view of Visual Basic because it is perceived as a simpler language to learn that others like Java and C++. The same students also view that Visual Basic has wider range practical applications especially in terms of the various connections it provided with the different databases and web browsers. However, business education students do not see it the same way. For these students, understanding the logic of the program is harder. Dealing with the syntax is more of issue to these students. They just do not view the time spent looking at a program to find simple syntax errors is a feasible use of time. At the same time, the most recent versions of Visual Basic stresses the use of Object Oriented Programming (OOP) techniques, so the same students have to learn these OOP principles on top of other programming concepts. This additional complexity translates into difficulty for these students learning how to program. So in general, business education students do not favor learning to program using the general purpose programming languages while, on the other hand, technology students favor it.

To simplify the comparison between the two programs, Table 1 below shows the information related to the selection of a programming language at both Institutions (IUP and RMU) as pertains to the factors listed in this section.

General Thoughts about Selecting a Language at the Two Programs

When choosing a language, one needs to consider why require computer languages at all. Alice allows the student to take everyday movements of simple objects and structure them to create a more general movement. VB.Net and other similar languages are designed to implement a structured solution for business oriented tasks. If logic development as an abstract concept is the goal, Alice certainly allows it to progress more easily and familiarly. If solving a business-oriented problem is the goal, VB is the more robust language for solving this kind of problem easily. Using Alice to solve business problems becomes very demanding and actually more difficult to implement. The program at Robert Morris is designed for the logical analysis and solution of business problems, and specifically leads to more advanced courses linking with files and the web. If students are expected to transfer learning to a more detailed course, they would have to relearn many of the syntactical concepts. It seems that for those who want to know about logical concepts in general, Alice is simpler, but for those who need to know about the level of specification required to solve business problems, VB would be a better fit.

SUMMARY

This paper discussed the issues and challenges that face the decision to select a programming language to teach for students enrolled in a master degree program in education. It began by explaining the factors that make learning programming languages a difficult task for students. It then introduced a programming language that is intended to provide solutions for the points of difficulty that accompany the learning of how to program. The paper then elaborated on the experience of two graduate degrees in education that teach programming in their respective programs. The Master of Business Education at Eberly College of Business at Indiana University of Pennsylvania and the Master of Science degree in Education at Robert Morris University. The first program uses

Alice to teach in their programming topics and the second one uses Visual Basic in their first programming course. More details on the factors that led each program to select the language they had selected for their programming course and the responses of the students was also discussed in the paper.

Table 1 – Comparing Programming Language Selection Criteria—IUP and RMU

Program Name	**IUP**	**RMU**
General Purpose Vs. Beginner programming	Beginner programming language (Alice)	General purpose programming (Visual Basic)
Major only course versus service course	Major in Business Education only course	Required by technology majors, business education majors as well as other majors
Entire Course Versus Selected Topics	One programming topic among four selected topics	Entire course teaches programming
Standalone Versus Prerequisites	Stand alone. Does not serve as a prerequisite to other courses	Though not formally a prerequisite, there is understanding that this a first course for other courses

REFERENCES

1. Adams, J. (2008). Alice in Action Computing Through Animation. Boston, Massachusetts: Course Technology.
2. Anewalt, K.(2008). "Making CS0 fun: an active learning approach using toys, games and Alice". Journal of Computing Sciences in Colleges, 23(3), 98-105. Retrieved March 28, 2008 from ACM Digital Library http://www.acm.org/dl.
3. Baldwin, L.P.; Kuljis, J. (2001). "Learning Programming Using Program Visualization Techniques". Proceedings of the 34th Hawaii International Conference on System Sciences – 2001. Retrieved April 17, 2008 from IEEE Computer Society Digital Library http://www.computer.org/portal/
4. Carter, J.; Jenkins, T. (2002). "Gender differences in programming?". Proceedings of the 7th annual conference on Innovation and technology in computer science education, Retrieved April 15, 2008 from ACM Digital Library http://www.acm.org/dl
5. Dann, W.; Copper, S. & Pausch, R. (2006). Learning to Program with Alice. Upper Saddle River, NJ: Prentice Hall.
6. Guibert, N.; Girard, P.; Guittet, L. (2004). "Example-based programming: a pertinent visual approach for learning to program". Proceedings of the working conference on Advanced visual interfaces 358 – 361. Retrieved March 30, 2008 from ACM Digital Library http://www.acm.org/dl
7. Herbert, Charles (2007). "An Introduction to Programming with Alice". Boston, Massachusetts: Course Technology.
8. Kelleher, C; Pausch, Randy (2005) "Lowering the Barriers to Programming: A Taxonomy of Programming Environment and Languages for Novice Programmers". ACM Computing Surveys 37(2) 83-137. Retrieved March 28, 2008 from ACM Digital Library http://www.acm.org/dl.
9. Marrero, W.; Settle, A. (2005). "Testing first: emphasizing testing in early programming courses". Proceedings of the 10th annual SIGCSE conference on Innovation and technology in computer science education, 4-8. Retrieved March 28, 2008 from ACM Digital Library http://www.acm.org/dl.
10. Porter, R.; Calder, P. (2004). "Patterns in learning to program: an experiment?". Proceedings of the sixth conference on Australasian computing education - Volume 30, 241-246. Retrieved April 18, 2008 from ACM Digital Library http://www.acm.org/dl
11. Powers, K.; Ecott, S.; & Hirshfield, L. (2007). "Through The Looking Glass: Teaching CS0 with Alice". ACM SIGCSE Bulletin 39(1) 213-217. Retrieved March 28, 2008 from ACM Digital Library http://www.acm.org/dl
12. Rosmaita, B. (2007). Making Service Learning Accessible to Computer Scientists.

SIGCSE 07, March 7-10, 2007. Retrieved December 20, 2007 from ACM Digital Library http://www.acm.org/dl