Issues and Challenges for Selecting a Programming Language in a Technology Update Course

Azad Ali
azad.ali@iup.edu

Scott Mensch
s.mensch@iup.edu

Technology Support and Training Department
Indiana University of Pennsylvania
Indiana, PA 15705, USA

Abstract

The purpose of this paper is to identify the issues and challenges that face the decision to select a programming language to teach in a technology update courses. This paper also makes suggestions to address these existing issues in an effort to increase student success. The findings of this paper has been implemented in a technology update course that is part of a master degree program in education within the Technology Support and Training (TST) department at Eberly College of Business and Information Technology (ECOBIT) at Indiana University of Pennsylvania (IUP).

Keywords: Introduction to Programming, Programming for teachers and educators, Programming for high schools, difficulty learning programming

1. INTRODUCTION

Educators in the computer field are facing a dilemma when updating the content of their technology courses. With enrollment dropping steadily in technology related courses in the past several years, efforts have been made to strengthen courses to boost enrollment. Among the technology area that has been subject to continuous revisions and updates are the teaching of programming languages. Learning to program is considered to be a difficult task for many students and remains to be contributing to the continuous drop in enrollment in computer related programs. It is estimated that between 25 to 80 percent of students drop their first computer classes due to the difficulty they face in learning a program language (Carter & Jenkins, 2002).

Despite the difficult points of learning to programming, teaching programming remains an integral part of pedagogy in technology update courses. Moreover, the issues surrounding the difficulty of programming may need to be addressed when teaching such programming languages. This paper outlines the experience of faculty within the TST department at IUP in updating their courses that contain aspects of programming. The TST department offers a master degree program in education (M.ED) and teaches a

course in technology update in the same program. The technology update course includes programming topic among a number of additional topics. This paper explains how the faculty within the TST department addressed the complexity that programming languages introduce when revising their technology update course.

2. PROGRAMMING LANGUAGES – DIFFICULTY POINTS

Learning the art of computer programming is considered to be a difficult task for many students. By the same token, teaching programming courses is found to be equally difficult for educators in different fields of study (Baldwin & Kuljis, 2001). In some cases, teaching programming is considered a "turning-away" factor from courses in programs that do not specifically teach technology related courses.

Numerous studies have been conducted to identify the factors that contribute to the difficulty in learning to program. Baldwin and Kulijas (2001) explained about the difficulty that students face when learning to program "The majority of students, even those enrolled in computer science courses, find computer programming a difficult and complex cognitive task" and further noted that "learning programming demands complex cognitive skills such as planning, reasoning and problem-solving" (p. 1).

Other studies provided more comprehensive analysis of the factors that contribute to this difficulty. Dann, Cooper and Pausch (2006) listed four factors that contribute to the difficulty in learning to program: 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 (2005) 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 one common fact and that is learning to program for a beginner is a 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 these reasons that contribute to the difficulty in learning to program.

Rigid Syntax and Commands

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 spoken languages. These differences have to do with the structure of commands, the stopping character, naming of variables, passing parameters and other related issues when structuring lines in programs.

A computer program is written to execute a command or a series of commands (Porter & Calder, 2004). A program contains a series of instructions that use a set of variables to perform specific tasks. The program is usually typed in a text editor and saved. It is then compiled to check for the correctness of the syntax. If there is an error in the program, the compiler displays an error message to tell the exact location and meaning of the error. The programmer accordingly fixes the errors, recompile and repeat the cycle again until all error are

fixed and the program will be ready to be executed.

A common programming language used to be the BASIC programming language. The syntax in BASIC was simpler, thus fitting beginners who take it as their first programming language. The commands were separated by lines, in which one line represented one command. The lines were numbered sequentially, thus following the commands used to take sequential or logical order, similar to reading a book. The commands were in a simple form that could be more understandable sometimes even to novice programmers.

BASIC was also simple enough to teach to beginning programmers. BASIC however was not able to handle large tasks that require writing longer programs. Thus the call increased to produce a language that is more understandable to the general user. The call for a language that uses "English Like" statements started to overcome this problem. COBOL was the language used because of this feature. COBOL uses commands that looks like or resembles spoken English language. However, programs written in COBOL were longer and the programmer had to type all these commands which then increased the chance for mistyping or misunderstanding the commands.

The use of the newer languages in the curriculum, such as Java and C++, added a new dimension to the complexity of the syntax. These languages attempted to minimize coding. They used a lot of abbreviated codes to make it easer to write programs. However, these languages used characters that are easy to miss while typing. For example, the program used characters such as a semi colon to indicate the end of the command and the use of curly brackets to indicate beginning and end of block of codes. These characters can be confusing and easily mixed with regular brackets. Also, users sometimes do not know when to use regular brackets or curly brackets because both types are used at various stages of within the program. These issues get complicated when the program contains multiple and nested block of codes. These blocks of code must each have their own opening and closing brackets. When these blocks of code are nested at multiple levels, beginning programmers have trouble understanding which bracket belongs to what block.

The error messages that are generated by the compiler are not always helpful. Sometime, these error messages are designed for advance programmers and the wording of the messages do not help beginning programmers understand their meanings. 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 programmer keeps looking at the line where the error message is pointing and can't identify the error which can become frustrating.

Unfamiliar Structure

Creating a structure is not new in academia. Structure has been used in different fields of fields of study and provides a number of advantages when used. 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 in 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 that are mentioned above.

The sequence control structure is where the commands are executed one line after another and is the simplest one to follow. Programmers can follow the code by reading one line after another. Although the commands may sound mysterious and the variables may not be clear, the lines can be read similar to reading a book. Familiarity with the syntax may help understand the program, but in either case it follows a pattern that is familiar to most people. Due to this it may be more understandable than other control structures.

The selection control structure is included when the program executes certain blocks of

code based on different conditions. The blocks of code may be within the same program file, or it can be written in a totally different program. During the execution of the program, branching out of sequence may lead to another location and then the program may encounter another selection statement that branches out to another location. This kind of branching out may continue at several levels and it may not be clear at what point the program goes back to the original code that it branched out from. Two difficult points arises her from this kind of branching out. First, identifying the block of code that is executed as a result of this branching out is difficult. Second, identifying the multiple level of block that are executed and then to return back to the original line of execution is tricky. Both of these difficulty points confuse many new programmers and lead to frustration as the program becomes longer.

The iteration, or loop control structure, faces similar problems but at a different level. In this structure, programming code is executed a specific number of times or until a specific condition is met to halt the execution of the loop. The difficulty here is similar to the selection statement in that it is not clear which statements are executed within the loop. Also, the idea of repetition is foreign to many users in thinking of a loop until a condition is met.

Initial teaching programs in BASIC programming language did not follow a particular structure. Programs were written in one block of code and they were not broken down into many distinct blocks. Beginners did not have to remember different module names, nor how to pass variables, parameters and other related names. This practice was simple enough to teach to students unfamiliar with programming. However, as programs became longer and as some of the tasks were repeated from one program to another the need to structure the solution increased. This structured solution is achieved with two purposes in mind: first to break down the program into smaller modules to make them easy to read and follow. The second purpose is to increase the usability of the code [12]. In other words, a module written for one purpose need not be repeated in other programs over and over again. Instead, the module is written independently and other programs use the same module in their programs. However, this multiple level of branching out and calling other programs is confusing.

This call for structure is designed to make a program development cycle more efficient and aimed at standardizing the coding of programs and reusing existing code of the programs. However, this kind of structure is difficult for inexperienced programmers. Later development in the programming industry introduced the use of Object Oriented Programming (OOP) methodology which stresses more of the usability and the structure issue in the program. The OOP methodology introduced many new concepts that needed to be understood along with learning the programming concepts. Dann et al [5] 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 places an extra layer which make it more difficult to learn.

Time/Output Ration

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 programming examples may include writing a program to convert Fahrenheit to Centigrade or converting miles driven to kilometers.

Writing the programs to produce the examples that are mentioned above may follow different steps when switching from one programming language to 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 the average students that do not know a lot about programming, putting this kind of effort to produce a simple output may not be justified and may not be time efficient. After all, the same students can repeat similar statements and produce the similar calculations multiple times with less effort. Students may question the feasibility of spending this

much time to produce simple outputs that are generated from writing programs.

3. PROGRAMMING LANGUAGES – SIMPLIFYING THE PROCESS

A number of studies have been conducted that acknowledged the difficulty with learning a new programming language and suggested steps to simplify this learning process. These suggestions range from simple (such as changing the programming language) to more detailed suggestions that deal with the conceptual model and the paradigm of teaching the programming languages.

In a study that explained about the conceptual model and the learner's understanding of the programming language, Baldwin and Kulijas (2001) noted the following:

> "It has been argued that conceptual models can serve to enhance learners' conceptual understanding of programming. The methods used to enhance the development of accurate mental models include: designing the interface so that users can interact actively with it; using metaphors and analogies to explain concepts; and using spatial relationships so that users can develop capabilities for mental simulations (P.1)

Dann, Cooper and Pausch (2006) noted three topics that students in programming courses should learn: algorithm thinking and expression, abstraction, and appreciation of elegance. Adams explained that in order to solve the problems associated with introductory programming courses, faculty must include examples that are engaging and capture the imagination of today's student.

Herbert (2007) explained that the best way to teach programming ideas is to expose it to the students gently, and then to gradually add more and more detail until one day they realize they've learned quite a bit. This kind of approach to learning programming is referred to as the "spiral approach". The process can be long and sometimes tedious, thus programming educators need to motivate students along the way to keep them interested.

Dealing with the Syntax

The programming industry in general has been trying to solve the syntax issues of programming languages for some time. Starting with the early days of teaching programming, error messages displayed from compilers were generally vague and did not help in identifying the meaning of the error or the location. As programming increased, the error messages became more descriptive and meaningful.

As programming turned into full gear with the Graphical User Interface (GUI) objects, more help was given to programmers. Manuals and online help provided examples of how to code. Description of error messages became more elaborate. Helpful hints were given as the programmer types the program. An example of this is the use of "intellicense" where the users type something and the system gives suggestions to complete the commands.

Despite these advancements, programmers in general still had to type the program. Additional studies suggested means of communicating instructions that did not include a lot of tying. Kelleher and Pausch (2005) suggested simplifying the syntax so that beginners can more easily learn or find alternate ways to communicate their instructions to the computers. Baldwin and Kuljis (2001) suggested the use of visual systems in creating a program and further noted about this issue:

Herbert (2007) noted that in order to make it easier to learn programming, three factors must be maintained: minimize the syntax, provide visual feedbacks and shorten the creative cycle of conceptualization, and improve the implementation and results. Baldwin and Kuljis (2001) on the other hand, stressed about the benefits of using visual metaphors because they provide more meaningful clues than verbal ones.

Syntax problems occur as a result of programmers incorrectly typing commands into the program. An effective way to deal this problem is to eliminate typing the

syntax. However commands have to be coded into the program in order for the program to be executed and produce the intended output. A number of studies have suggested using visual objects that produce commands and lines of code. These visual objects can be buttons or images that can be dragged and dropped into the lines of code. Once dragged, the objects provide the programmer with different options. For example, if programmers want to execute moving an object from one location to another, the can do so by dragging the object from the location where it is placed move and drop it in code. Once dropped in the correct location the program then displays a menu asking for the next option to select. In other words, this kind of coding eliminates the possibility of syntax errors by using predefined visual objects that can be moved and placed in the lines of code.

Addressing the Structure Issue

The structure of the program has been addressed in various programming languages. Programming code editors have been improved significantly and serve as an aid to help the programmer with the structure. Some editors require the programmer to indent certain elements of the program to indicate belonging to a particular block or segment of the program. Other editors do the indentation for the programmer. An example of this indentation would be coding an "if" statement in the program. Once the programmer codes the first line of the statement, the editor automatically indents the code under it to indicate a subordinate. Other editors complete the statement for the programmer such as the case in writing HTML programs. When the programmer codes the opening tag for HTML, the editor automatically completes the code for the closing tag once the user types the back slash character.

The use of flowcharts, pseudo code and hierarchy charts was introduced to illustrate the different segments within the program and the relationship among the segments of the program. In these tools, programmers draw symbols that show these segments. Introducing objects to help with program code rather than tying the commands provides an advantage in understanding the structure of the program. Providing these "visual" objects make the concept more concrete because these provide visual clues as compared to the same display when it is typed.

Enhancing Time/Output Ratio

Since programming is considered "boring" by many accounts, providing some interesting applications may help fade away this notion. The output that is generated by a program may be more interesting if it provides visual clues such as photos, shapes, buttons and so on.

Engaging the user in the program output helps with the development of the program at several fronts. First it provides a feedback to the user, and second it helps makes programming interesting (Porter & Calder, 2004). The introduction of GUI objects helped with this type of output when writing programs. By using GUI objects, programmers can use buttons, shapes and other tools that make a program more interesting than just plain "text" output.

Guibert, Girard and Guittet (2004) stressed on the positive experience of using programming by example (PbE) where programmers design methods to provide continuous feedback. In other words, such programs contain methods that are intended to continuously provide feedbacks to the programmer about the program execution, thus engaging the programmer during program execution.

4. ALICE PROGRAMMING

Alice programming 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 cuts down on the syntax and coding.

Alice has increased in popularity in 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 (2008) noted 3 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.

Object-oriented programming. Alice includes a huge library of off-the-shelf 3D objects ranging from astronauts to ants, cowboys to castles, fairies to farms, mummies to motorboats, ponds to pagodas, robots to rowboats, skyscrapers to space shuttles, turtles to T-rexes, wizards to waterfalls and zombies to Zambonis – each of which can be animated through a variety of predefined methods. Alice makes t easy to build 3D worlds from these objects. Those objects can be animated using object-oriented programming.

The remainder of this section details how the Alice programming language addressed the difficulty factors that were identified earlier.

Alice Syntax

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 (Powers, Ecott,& Hirshfield, 2007). 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.

The Structure in Alice

Alice uses the structure of object oriented programming (Marrero & Settle, 2005). Alice uses classes of objects and each object has its own properties and methods. Classes are clearly defined through the use of the 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, and events. It passes parameters, receives output, and creates a structure to the program. All of this is done very similarly to other programming languages except that Alice uses visual objects which are easier seen and understood.

The Output from Alice

Alice uses visual output. All objects within Alice are three dimension visual objects, thus the output that is generated from a typical Alice program 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 in Alice program 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 the 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.

Drawback of Alice

The main drawback of teaching Alice is that it is strictly used for educational purposes

and for beginner programming cases. In other words, people who learn Alice do not expect to use it in payroll applications, inventory or scientific applications, or business environments. Instead, teaching Alice is strictly used for teaching introductory concepts of programming. Newer versions of Alice can be easily lined to other programming languages such as Java, so it may help transitioning into more advanced programming courses. But generally, Alice is used for teaching in introductory programming courses.

5. THE TST PROGRAM AT IUP

The TST department within the ECOBIT at IUP offers two bachelor degrees and one associate degree. The first Bachelor of Science degree is in Business Technology Support while the second is in Business Education. The Associate degree is in Computer and Information Technology.

The TST program also offers a master degree in business education. 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 dilemmas the course must begin by teaching the principles of programming and the syntax and logic of programming.

The Alice programming language was selected for this course. The main reason for selecting Alice is that the students in this course are perspective 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 feedback from selecting Alice in this course has been positive and enrollment has increased in this course since introducing Alice to the course. 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.

6. 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 in education. It focused on the factors that make learning programming languages a difficult task for students and the steps that have been taken to simplify the teaching of programming languages. The paper further elaborated on the new language that is increasingly being taught in introductory level programming courses: Alice. In addition this paper focused on the experience of a course that is taught at the Technology Support and Training (TST) program at Indiana University of Pennsylvania. This course introduced Alice as the programming language to teach and feedback from students enrolled in the course has been positive about the selection of Alice

7. REFERENCES

Adams, J. (2008). Alice in Action Computing Through Animation. Boston, Massachusetts: Course Technology.

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.

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/

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

Dann, W.; Copper, S. & Pausch, R. (2006). Learning to Program with Alice. Upper Saddle River, NJ: Prentice Hall.

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

Herbert, Charles (2007). "An Introduction to Programming with Alice". Boston, Massachusetts: Course Technology.

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.

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.

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

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