Cite as: Ali, A., & Smith, D. (2014). Teaching an introductory programming language in a general education course. *Journal of Information Technology Education: Innovations in Practice, 13,* 57-67. Retrieved from http://www.jite.org/documents/Vol13/JITEv13IIPp057-067Ali0496.pdf

Teaching an Introductory Programming Language in a General Education Course

Azad Ali and David Smith
Indiana University of Pennsylvania, Indiana, PA, USA

azadali@iup.edu; david.smith@iup.edu

Abstract

A department of computer science (CS) has faced a peculiar situation regarding their selection of introductory programming course. This course is a required course for the students enrolled in the CS program and is a prerequisite to their other advanced programming courses. At the same time, the course can be considered a general education course and may be taken by students from other majors as well. Both student populations require the department to teach the course at different levels of depth. CS students need it be covered in more depth to prepare them for their upper level programming courses. At the same time, students from other majors who are taking it as part of their general education course will not be interested in this level of depth. Added to this is the fact that taking a first programming course is considered difficult to most students. Thus many factors are considered for the selection of a programming language for this course. After further experience and additional technological development, the department redesigned their course and curriculum in order to provide most effective solution to this dilemma. The experience of this department in reaching this solution along with the relevant literature reviews are discussed in this paper.

Keywords: Programming and liberal studies course, Programming for non-majors, Programming general education course

Introduction

General Education Courses are called "general" because of the nature of the content covered in them, the depth of knowledge required, and the diverse student population that typically are enrolled in them (Gordon & Steele, 2003; Nelson Laird & Garver, 2010). Having students from different majors/backgrounds in these courses brings challenges to their teaching. These challenges include deciding the depth required in the course and how to capture the interest of students whose minds are already set on a different major (Pregitzer & Clements, 2013). These challenges become more complicated by two other factors: first, the course is considered difficult and thus a source of attrition for students enrolled in the major. And second, the course is a pre-requisite for additional advanced courses in the major for the teaching department. In these cases, design of the course may need to be altered to accommodate these factors. The introductory computer programming courses has these two factors: first, they are considered difficult by many students (Ali & Shubra, 2010; Daly, 2011; Kaplan,

Editor: Lynn Jeffrey

2010); second, a first programming course is typically a prerequisite to other advanced programming and other major courses. Figure 1 below depicts these elements.

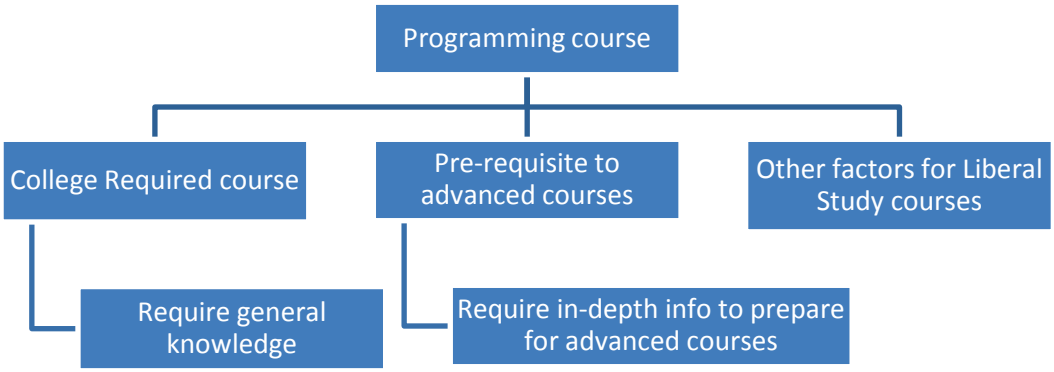

Figure 1 - Factors in Designing First Programming Course

The Department of Computer Science (CS) at Indiana University of Pennsylvania (IUP) was faced with this situation. The CS department teaches a programming course that is a general education course that requires learning a first computer programming language. Learning to program one's first computer programming language is considered to be a difficult task even for students enrolled in IT related majors. For other students, it is even more than that: it is a source of attrition from the course or even the program. The department went through different iterations of course design and came up with a course design to accommodate these factors. The experience of this department in researching these issues and modifying the design of the course is illustrated in this paper.

Study Outline

The remainder of this paper is divided into the following five sections:

- The first section reviews literature and explains general education courses, their characteristics and how they work within the educational system.
- The second section discusses the factors that make learning to program a computer and taking first programming courses a difficult task.
- The third section explains the characteristics of a programming language called "Alice" that addresses many of the issues associated with the difficulty of learning to program.
- The fourth section explains the experience of the CS department and how they modified the design of the course(s) to accommodate the factors explained above.
- A summary of this paper and suggestion for future follow-up study is introduced at the end of the paper.

General Education Courses

A **general education** curriculum, by definition, is wide-ranging in its scope of topics, disciplines, and applications. However, many students enter college with specific personal interests or affinities for particular areas of academic study. This interest gap between individual student interests and **general education** offerings is frequently expressed by students in their desire to "get the core **courses** out of the way." For those institutions that appreciate the role that a **general education** curriculum plays in providing students with a holistic liberal arts **educa-**

> **tion**, this "get them out of the way" attitude must be addressed. Through the application of text-based situational interest research, and creative writing principles and instructional design, this article offers educators theoretical insights and practical ways to stimulate student interest in online **general education courses.** (Pregitzer & Clemente, 2013 p. 162)

General Education Courses (GEC) typically are required courses for all the students enrolled at a given university or a college. The list of these courses can cover different departments within the college or university (Robertson, 2013). GEC courses are typically taken by the students early in their enrollment in academic studies. The main goal of offering GEC is "to provide broad knowledge and skills base for all college graduates, independent of their major" (Offerdahl & Impey, 2012, p. 19).

GEC brings advantages to the teaching departments because they give them courses to teach (Gardiner-Shires & Heinerichs, 2012). Aside from teaching the content of the courses, GEC is useful in two ways: First, they can serve as a marketing tool for the teaching department to bring more students into their programs. Second, GEC can also serve as a pre-requisite for other courses within the department and prepare student for advanced courses in the program. However, teaching GEC often works against the department especially when the course is considered a source of drop out or attrition for their programs. The design of the course may play a part in each of the factors listed above. The remainder of this section explains how each of the three factors plays in terms of teaching GEC courses.

GEC as A Marketing Tool for the Teaching Department

Pregitzer and Clemente (2013) noted that the majority of students entering colleges have already made up their mind about the major they want to be enrolled in and little can be done to change that. However, the number of students that are undecided about their majors may be significant as well. Gordon and Steel (2003) conducted a 25-year study in which they conclude this number to be between 18% – 20%, inclusive of those with leanings toward one major or another. For these students the courses they take early during their exploration of the GEC may influence the selection of their major.

In a study conducted to assess the knowledge of high school graduates who are entering college (Daempfle, 2003) it was found that most students obtained their first knowledge about a subject during their first year of college. Although the selection of majors by students goes into different factors beyond of the scope of this paper, the experience in the first course of a particular subject cannot be discounted. The first experience of a topic is typically encountered in the GEC. A favorable experience in the course may persuade the undecided students into "deciding" on a major. It could even persuade students who are not firmly decided on a major to switch into another based on this experience in GEC.

GEC as Potential Drawbacks to the Teaching Department

Similar to the advantages that GEC can bring to the teaching department, GEC can also be a source of drawbacks. In particular, if the GEC course being taught is considered difficult, strange. or just does not fit the interest of the students, this may lead to a higher attrition rate within the course and lower retention among the students in the major. Math, science, and technology are some of the fields that experience high attrition rates among their students (Daempfle, 2003; Thompson & Bolin, 2011). Computer programming courses fall into this category as well. In some cases, at least 25% of students dropped out of their technology majors after taking their first programming courses (Carter & Jenkins, 2002; Kelleher & Pausch, 2005).

Studying attrition in different programs can lead to wide range of discussion and analysis that goes beyond the scope of this paper. However, the impact of the first course experience for the students cannot be minimized. This first course experience for most students takes place during the time when they take their GEC. In other words, GEC can influence the extent to which the students like or dislike the course and hence decide to continue or drop out of the program all together.

A Pre-requisite for Advanced Courses

Introductory GECs are often considered a pre-requisite for advanced level courses for the students enrolled in the program of the teaching department. A course that is a pre-requisite to other advanced courses provides a certain depth in the first course to prepare the students for the advanced courses. If this is a pre-requisite for more than one course, then certain coverage is required to allow for easier transition by students to the other advanced courses. Coverage of this depth may go beyond the intended general content of GEC and may upset the non-majors because they do not relate to it similarly to what they have in the courses of their majors.

Abou-Sayf (2008) conducted a study to assess whether teaching a pre-requisite is necessary for the success of students in upper level courses. It studied the difference in performance in the upper level courses between students who took the pre-requisite and those who did not. The students who completed the pre-requisite prior to the course had the advantage, although the differences in performance were minor.

Pre-requisites for computer programming courses may take special direction. In the cases where an entry level programming course is required, it may or may not help students in upper level course. After all, most programming courses begin by discussing syntax and general structure of programming. Then the level required goes from that point in different directions. However, the first programming language intends to give the students a general sense of what can be accomplished in programming. They are not meant to give deeper or advanced level of applying programming methodologies and applications.

Difficulty in Learning First Programming Courses

> Coding the syntax of today's popular programming languages can be frustrating for students who are new to programming. To write a simple program in Java for example, you must add extra code just to get the program to print the famous "Hello World" to the screen. The program must be typed with the proper capitalization, pristine spelling, and the exact amount of curly braces, quotes, semicolons, brackets, and parenthesis in the correct order. How can we expect students to gain confidence with programming, if they are constantly receiving compiler errors that are preventing them from executing their programs? (Daly, p. 23)

The statement above echoes the prevailing sentiment in many programs that learning to program computer languages is considered to be difficult to most students. It is a source of trouble to the students enrolled in CS programs because it affects enrollment and retention for the program (Dann, Cooper, & Pausch, 2006).

Many students do not even consider enrolling in CS programs because of the perceptions about programming courses, some of which was echoed above. Computer programming courses are simply viewed as students sitting in front of a computer, glaring at the monitor, locating and correcting computer programming errors (Rosamita, 2007).

This difficulty with learning to program is worse for students majoring in other majors. It not only pushes the students away from CS majors, but it also has them take alternative courses or courses from other institutions (Tillberg & Cohoon, 2005).

Some may think that limiting the content of the course may solve the problem. After all, if the tasks will be limited to producing simple computer output without getting into the details of looping and advanced topics the problem may be solved. However, the power of computer programming will not be realized unless the students learn these advanced topics. Limiting the content may make students think that they spent an entire course learning something very trivial, as in the case of "Hello World" (Westfall, 2001).

Four reasons were cited for the difficulty to learn to program: rigid syntax, unfamiliar structure, length of time to develop a program, and working in isolation.

The syntax for computer programming is considered "rigid" because it follows rigid rules that do not allow for maneuver and deviation. The rigidity is clearly highlighted when using characters like colon, semi-colon, curly brackets, and others when coding programs. If these rules are not followed correctly, the compilers give what are called "syntax errors". To an inexperienced person these error messages may not appear helpful. Furthermore, a syntax error may be reported at a location within the program that may be many lines away from the source of the error. Overall, this frustrates the coder and, for the students, it may result in them dropping the program all together (Porter & Calder, 2004).

The structure of programming is unlike the structure followed in any other field of study. The general structure that is followed in programming (sequence, selection, and iteration) cannot be compared to general or typical examples in life like building a house, taking a trip, or investing money. Instead, these structures have their own methodologies that require different thinking pattern in order to be understood (Kelleher & Pausch, 2005). The continuous work on this kind of unfamiliar structure eventually leads to frustration for the students. The end result is that the students drop out of the program after taking the first programming course.

The time it takes to complete and successfully execute a program is long in comparison to other tasks. Displaying simple messages on the screen or performing simple calculations (like changing miles to kilometers) is not considered a task worthy of writing multiple lines of code to achieve. This can be more problematic if it is compared with the simplicity of their completion in real life. Doing simple calculations can be accomplished using simple calculator and displaying a message can be done easily. Unfortunately, these examples are often used in many first programming courses (and hence in GEC if a programming course is taught in the GEC).

Working in isolation is another factor perceived by students as typical for students taking programming courses. The common perception is that programming courses have students spend hours sitting in front of a computer – alone – trying to figure out what is wrong with the program they are writing. This perception, especially in GEC, may lead students to see or take courses other than those that require programming.

Teaching Alice in Introductory Programming Courses

> The Alice environment eliminates the frustration and focus on the syntax of the language, making it easier on students to create animation and/or games. Alice allows the programmer to create code without worrying about semicolons, curly braces, etc. and allows the students to focus on the concepts. The software not only impresses the students into a programming rich multimedia environment in which they feel comfortable manipulating and programming objects, but it truly enforces the object-oriented concepts needed if students are thinking about taking their programming to the next level. (Daly, 2013, p.24)

Learning about the factors that makes learning to program a difficult task may raise the question of dealing with these difficulties, especially if a programming course is selected as a GEC. Fortunately, various efforts were spent in order to make programming easier in first programming courses (Anewalt, 2008; Daly, 2011). Some of these efforts culminated in the development of a programming language that is intended to address these particular issues of the difficulty of learning to program computer languages. The name of this language is Alice. Adams (2008) notes the advantages to using Alice in introductory programming courses: absence of syntax, an Integrated Development Environment (IDE) which allows for using familiar well known structure and examples, and a game-like environment which allows for more interaction during program development.

This section explains the use of Alice in introductory programming courses. 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. It then explains the drawback of selecting Alice as the language to teach in such courses.

About Alice Programming Language

Alice is a programming language that was introduced by Carnegie Mellon University to address many of the issues raised concerning the difficulty of programming languages. Alice provides a visual interface that makes it easier to follow, and it cuts down on the time need for syntax and coding (Herbert, 2007).

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.

Advantages of Teaching Alice as First Programming Course

The main advantage of teaching Alice as a first programming course is that it addresses all the four issues that make learning to program a difficult task.

First, the issue with syntax is resolved in Alice. There are no syntax problems in Alice. Instead, using Alice as a programming language, 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.

Second, programming in Alice enables individuals to see the results at any time during development and at the completion of their programs (Adams, 2008). As the objects are pulled off the visual library, the programmer can elect to run their partially completed program and observe the effects. As a result, programmers using Alice are able to literally “see” their programs action during development and at completion of the program.

The third addresses the lack of motivation in learning to program which is viewed as a reason for pushing students away from programming courses (Daly, 2011). This 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 programming resolves this as well. Alice uses visual output. All objects within Alice are three dimensional visual objects. This output is visually appealing to most students. 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.

Fourth, development time for Alice programs is minimal compared to other popular 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 (as the case to practices in learning other popular programming languages). 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 to 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 (Guibert, Girard, & Guittet, 2003). This all helps with the notion of "programming in isolation" because the program essentially engages the student.

Drawbacks to Teaching Alice for first Programming Course

The main drawback of teaching Alice is that it (Alice Programming Language) is strictly a teaching/learning tool. It is not used in the industry in applications like payroll, accounts receivable, inventory control or any similar applications. In other words, learning Alice may not get anyone a job at entry level programming in the industry. Students majoring in computer science or similar majors may not find this language very challenging or interesting. Some students may think that taking Alice will take away from them learning another general programming course (Adams, 2008).

Additionally, Alice does not address the difficulties that arise in a course using standard programming languages (Like Java or Visual Basic). Students have to face (sooner or later) the issue of syntax and the obscure logic. They also have to spend a longer time on developing programs as they progress into taking more advanced course and they may have to often work in isolation. So basically, what sounds like an all good plan for using Alice may put the students at a disadvantage in later advanced courses for the computer science majors.

Putting Alice on a student's resume may not contribute significantly to increasing the marketability of the student in particular if the student is selecting programming as a career. Most employers look for keywords when they advertise for jobs (Scott-Bracey, 2013). Taking a full course in Alice may deprive the students from claiming an industry programming language (hence marketable keyword) like C++, Java, Visual Basic, SQL or any other languages to their resume. Adding marketable keywords to resumes are important for graduating students because employers often search for these keywords when deciding to hire individuals for jobs (McLean, 2006). Employers often search resumes for these kinds of keywords, if they do not find them, the resumes are automatically excluded.

Teaching a Programming Course as GEC

The department of Computer Science (CS) at Indiana University of Pennsylvania (IUP) teaches an introductory level course that is taken by students enrolled in their CS major but can also be considered as a GEC for the students enrolled in the College of Natural Science and Math at the same university. The course COSC110 which uses the C programming language worked as the only introductory programming course for majors and GEC for a number of years. However, recent technological advances and the difficulty experienced by students learning C programming language required the department to make changes to this course and to their curriculum. The remainder of this section explains the changes that made by this department to their COSC110 course in order to fulfill the requirements for their major as well as those for GEC.

COSC110 as the Only Course

With this approach, C programming language is taught as a course for CS students and other students taking this course as part of their GEC. This helped CS students because C programming is considered a pre-requisite for the advanced COSC210 Java course. However, it did not help the other students due to the difficulty associated with learning to program using general programming languages (like C language). The department faced significant attrition among students in their majors after taking this course. The course also faced a large number of students dropping out of this course prior to completing it.

Teaching Alice Alone in COSC110

With this approach, the Alice programming language is supposed to be taught for an entire course. This simplified the GEC course but it did not help the students enrolled in CS because this course serves as a pre-requisite for the other advanced programming courses. In particular, the claim was that Alice does not prepare the students well to take the advanced COSCS 210 Java course. Thus an alternate solution is sought to accommodate this issue.

Latest Decision – Two Courses

The latest decision reached by the department is to teach two courses: The first course COSC108 for GEC and COSC110 for CS majors, although CS students can take COSC108 and it can be considered for their required courses. At the same time, students from other majors can take COSC110 and it can be considered as GEC. However, the general notion is that COSC108 is for GEC and COSC110 is for CS majors.

COSC108 - Intro to Programming via Alice. Although the tile of the courses may imply teaching Alice only, the course is taught in two segments. The first half teaches Alice programming and the second half transitions into teaching Java with Alice. This course is more suitable as a GEC because it does not start the general programming topics from the beginning. As noted earlier in this paper, these are deemed difficult to students not majoring in CS. Instead, the course starts teaching Alice Programming Language. It makes it easier to understand the concepts of programming using objects and metaphors available in Alice. The second half of the course transitions the students into learning Java programming language using the interface they learned from Alice. Java is introduced to make it easier for the students to take COSC210 - Object Oriented & GUI Program course.

COSC110 - Problem Solving & Structured Programming teaches introductory Java programming. Java replaced the old C programming language in this course. The change to Java was made to make it easier for students to take COSC210. Having students with a solid background in the logic of programming along with learning the syntax of Java makes it easier to transition into the higher level COSC210 that teaches advanced Object Oriented programming. Figure 2 below shows how the department teaches programming language taking into consideration that it is a GEC, a course required for their major and a pre-requisite for the advanced COSC210 course.

Both courses equally prepare the students for taking COSC210 but with different approaches. COSC108 provides a solid understanding of object oriented terminology from the beginning, making it easier to understand the application of these concepts in the advanced course. COSC110 gets the students to learn more about the logic and syntax of Java to make it simpler to understand the application of the logic and syntax in the advanced COSC210 course in Object Oriented programming.

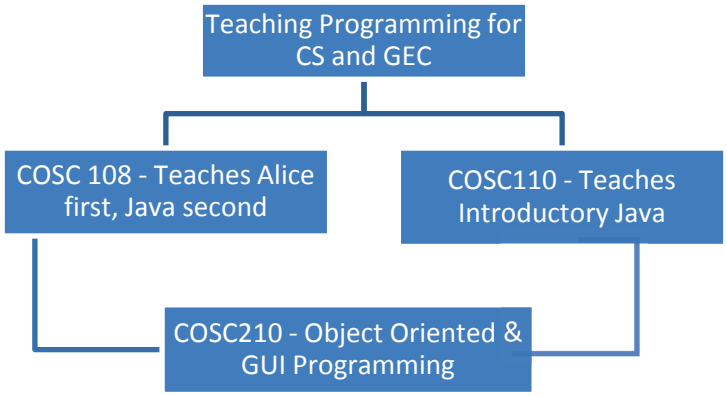

Figure 2 - Programming as GEC and for CS Majors

Summary and Suggested Future Research

This paper discusses teaching a general education course for programming. It began by explaining general education courses (GEC) and their coverage, the advantages they bring to the teaching department and the potential drawbacks to the same teaching department. The paper then explained about the factors that make learning to program a difficult task and delved into the discussion of Alice programming language that seems to provide answers to the issues that made programming such a difficult task. Lastly the paper elaborated on the experience of the Computer Science department at IUP that faced similar problems with their general education course in programming. It explained the different stages the department went through.

Although the paper explained the third stage, that is teaching both Alice and Java in the same course, it is a new experience to the department and the effectiveness of this later approach is yet to be determined. The intention of the authors is to conduct a study assessing the effectiveness of this last approach and comparing it to the other two approaches that were followed in the years before.

References

Abou-Sayf, F. K. (2008). Does the elimination of prerequisites affect enrollment and success? *Community College Review, 36*(1), 47-62.

Adams, J. (2008). *Alice in action: Computing through animation*. Boston, Massachusetts: Course Technology.

Ali, A., & Shubra, C. (2010). Efforts to reverse the trend of enrollment decline in computer science programs: A case study. *Issues in Informing Science and Information Technology, 7*, 209-224. Retrieved from http://iisit.org/Vol7/IISITv7p209-224Ali825.pdf

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://dl.acm.org/citation.cfm?id=1295109.1295133&coll=DL&dl=ACM&CFID=461783052&CFTOKEN=26182186

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

Daly, T. (2011, May). Minimizing to maximize: An initial attempt at teaching introductory programming using Alice. *Journal of Computer Science in Colleges, 26*(5), 23-30.

Daempfle, P. A. (2003). An analysis of the high attrition rates among first year college science, math, and engineering majors. *Journal of College Student Retention: Research, Theory and Practice, 5*(1), 37-52.

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

Gardiner-Shires, A., & Heinerichs, S. (2012). Promoting athletic training through a general education course in psychosocial aspects of sports injuries. *Athletic Training Education Journal, 7*(2).

Gordon, V. N., & Steele, G. E. (2003). Undecided first-year students: A 25-year longitudinal study. *Journal of the First-Year Experience & Students in Transition, 15*(1), 19-38.

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, C. (2007). *An introduction to programming with Alice*. Boston, Massachusetts: Course Technology.

Kaplan, R. (2010). Choosing a first programming language. *Proceedings of the 2010 ACM Conference on Information Technology Education*, 163-164.

Kelleher, C., & Pausch, R. (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

McLean, C. (2006). A foot in the door: IT job-search strategies. *Certification Magazine, 8*(4), 38-40.

Nelson Laird, T. F., & Garver, A. K. (2010). The effect of teaching general education courses on deep approaches to learning: How disciplinary context matters. *Research in Higher Education, 51*(3), 248-265. doi:10.1007/s11162-009-9154-7

Offerdahl, E., & Impey, C. (2012). Assessing general education science courses: A portfolio approach. *Journal of College Science Teaching, 41*(5), 19-25.

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

Pregitzer, M., & Clements, S. (2013). Bored with the core: Stimulating student interest in online general education. *Educational Media International, 50*(3), 162-176.

Robertson, S. N. (2013). General education: Charting a roadmap toward student success. *Peer Review, 15*(2), 18-19.

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

Scott-Bracey, P. (2013). Analyzing Internet job advertisements to compare IT employer soft skill demand versus undergraduate IT program curriculum programs in Texas. Retrieved 10/15/2013 from http://www.editlib.org/noaccess/38669

Thompson, R., & Bolin, G. (2011). Indicators of success in STEM majors: A cohort study. *Journal of College Admission, 212*, 18-24.

Tillberg, H. K., & Cohoon, J. (2005). Attracting women to the CS major. *Frontiers: A Journal of Women Studies, 26*(1), 126-140.

Westfall, R. (2001). Technical opinion: Hello, world considered harmful. *Communications of the ACM, 44*(10), 129-130.

Biographies

Azad Ali, D.Sc., Professor of Information Technology at Eberly College of Business – Indiana University of Pennsylvania – has 30 years of combined experience in areas of financial and information systems. He holds a bachelor degree in Business Administration from the University of Baghdad, an M.B. A. from Indiana University of Pennsylvania, an M.P.A. from the University of Pittsburgh, and a Doctorate of Science in Communications and Information Systems from Robert Morris University. Dr. Ali's research interests include service learning projects, web design tools, dealing with isolation in doctoral programs, and curriculum.

David T. Smith, Ph.D., Associate Professor of Computer Science – Indiana University of Pennsylvania – has 12 years experience in academia and 21 years of industry experience in database systems, computer language development, and other systems programming. He holds a bachelor degree in Physics and Mathematics Education from Indiana University of Pennsylvania, an M.S. in Computer Science from University of Central Florida, and a Ph.D. in Computer Science from Nova Southeastern University. Dr. Smith is active in consultancy and has research interests in artificial intelligence, distributed object computing, data mining, and software engineering.