Journal of Information Technology Education: Innovations in Practice Volume 13, 2014

Cite as: Ali, A., & Smith, D. (2014). A debate over the teaching of a legacy programming language in an information technology (IT) program. *Journal of Information Technology Education: Innovations in Practice, 13,*111-127. Retrieved from http://www.jite.org/documents/Vol13/JITEv13IIPp111-127Ali0773.pdf

A Debate over the Teaching of a Legacy Programming Language in an Information Technology (IT) Program

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

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

Abstract

This paper presents a debate between two faculty members regarding the teaching of the legacy programming course (COBOL) in a Computer Science (CS) program. Among the two faculty members, one calls for the continuation of teaching this language and the other calls for replacing it with another modern language. Although CS programs are notorious for continuous updates (and hence debates over the updates) of their courses, but the teaching of COBOL programming language has sparked many debates that have been on-going for years. Each side of the debate provides evidences that support their position.

This study, although, provides a debate over the same topic but it is different from ongoing debates because it balances the views expressed by both sides of the debates. This in turn shows the difficulty encountered by various departments when making a decision about COBOL. We are presenting in the paper a thorough literature review regarding both sides of views. We are also incorporating in this debate the lessons we learned from our long experience in this field. Thus, we will present our conclusion and recommendation in this paper based on the literature review and on our experience in this field. Our goal from writing this paper is two faceted: first, to present opinions about each side of the position regarding the teaching of COBOL, and, second, to reach a consensus regarding the continuation of teaching this programming language (or for this matter replacing it with another language).

Keywords: Teaching legacy language, Legacy programming language, legacy applications, COBOL

Introduction

> "The crisis in college and university COBOL education is becoming more acute as time passes. The reduction and total elimination of COBOL courses, declining interest by both faculty and students, and low visibility of new commercial applications being developed using COBOL are commonplace. Yet many practitioners are well aware of the billions of lines of existing, operational, revenue-producing lines of COBOL code that continue to provide the mainstay of many commercial enterprises" (Roggio, Comer, & Brauda, 2013, p. 1).

Editor: Peter Blakey
Submitted: 21 July 2014; Revised: 19 Nov 2014; Accepted 1 December 2014

The above quotation represents a common dilemma (and hence a reason to continuing debate) among many Computer Science (CS) programs: To teach the legacy programming language COBOL in a programming course or to replace it with another language? This debate has been ongoing in the internal communication of CS departments and also in CS literature. Some CS educators advocate that technological advances made the content and the use of COBOL in any course a thing of the past (Dunn & Lingersfelt, 2005; Hurwitz, 2011; KVSN & Choudhari, 2013). Yet, in many other programs, COBOL is still taught and there are experts who advocate keeping and teaching this programming language (Duckett, 2013; HFC, 2014; Whitson, 2007).

This debate has been ongoing at the department of computer science (COSC) at Indiana University of Pennsylvania (IUP). The COSC department has taught COBOL since its inception in 1971. The two authors of this paper represent opposite sides of the debate. The first author calls for eliminating COBOL and replacing it with Visual Basic. The second author advocates the continuation of teaching COBOL. Both authors decided to review the literature on this topic, to study the issues and contributing factors regarding both sides of the debate, and then to present their findings in this paper. Presented herein are the authors' findings along with their analysis and recommendation.

Study Contribution

Our contribution in this study is represented in the following points:

- The factors that lead to the difficulty (and hence the continuation of the debate) in making a decision for continuing or replacing COBOL in programming courses. This involves understanding a number of historical and technical factors presented in this paper.
- A balanced view of the features that stand for teaching COBOL and others that stand for deleting it (or replacing it with another programming language). This balanced view reflects the current debate in many academic departments teaching COBOL.
- Good information that helps in making a decision regarding continuing to teach COBOL or deciding against it

Study Outline

The remainder of this study is divided into the following sections:

- The first section explains the meaning, uses, and application of the term "legacy" in general and then in particular in relation to the information technology (IT) field. This paves the way to explaining the reasons for COBOL still being taught despite the label of "legacy" that is often attached to it.
- The second section gives a historical perspective on the selection of programming languages to teach. It describes how programming languages were previously selected and at the current time. It also explains the factors that influence such selection in both periods of time.
- The next two sections describe the arguments for (third section) and against (fourth section) teaching COBOL.
- The next section presents the argument made by both authors in regarding to teaching COBOL or replacing it with Visual Basic.
- The last section presents conclusions and recommendations regarding the teaching of COBOL.

About Legacy and Legacy Information System

More than any other programming language, COBOL is associated with the word *legacy* when talking about older technologies or older programming languages (Bisbal, Lawless, Wu, & Grim-

son, 1999; Comella-Darda, Wallnau, Seacord, & Robert, 2000). There are different reasons for this strong association between the two words "COBOL" and "legacy". The word "legacy" is used differently in general life than in the IT field. We will also discuss factors that lead to this strong association. This association has made a strong contribution to the difficulty of making a decisive conclusion regarding the teaching of COBOL.

The remainder of this section discusses the term "legacy", then goes on to explain the term 'Legacy Information System' (LIS), a system that is considered the cornerstone COBOL use in the industry. Last, legacy programming languages and how COBOL is distinguished as "the legacy" programming language is discussed.

Meaning and Terms of Legacy

The term "Legacy" is used widely in everyday language. It is defined in Webster's dictionary as "something (such as property or money) that is received from someone who has died" and "something that happened in the past or that comes from someone in the past." It is also used extensively in the Information Technology (IT) field literature as well.

The use of the term "legacy" was extended to the IT field to characterize older or unused technology. Webster's dictionary defined legacy in terms of computer and technology system as "relating to, or being a previous outdated computer system" (Legacy, 2014). The word "legacy" has gained greater currency in technology due to the rapid changes in technology (Gabriel & Steel, 2008). Thus older software and technology is often described as legacy. For example, the first computer "ENIAC", now obsolete, is called a legacy technology. More recently, old systems such as DEC, Wang, and Honeywell, have been called legacy. The use of punch cards to enter data is known as a legacy data entry system. Legacy can describe older software tools that are currently not in use. Take, for example, the use of DBase 2 or DBase 3, WordStar or similar software tools. These were popular at some time in the not too distant past, but are rarely in use these days. Thus, these tools are also called legacy technology (Laguna & Crespo, 2013).

The American Programmer web site (http://theamericanprogrammer.com/) explains the use of 'legacy' as "bequeathed by someone who died". Although this definition likened it to death of a person, this sentiment is often mentioned in reference to technology that is dead (we mean here as being eliminated or not being used at present time). It can also reference a technology that is near its death on its way to being eliminated.

Legacy Information System

The general understanding of Legacy Information System (LIS) is that it is an information system that uses old (legacy) technology in terms of software and/or hardware. But the analogy also references a technology that is riddled with problems and impedes progress or movement toward newer technology. Paradauskas and Laurikaitis (2006), for example, defined LIS as "any information system that significantly resists modification and evolution to meet new and constantly changing business requirements" (p. 214).

Despite the wide association of LIS with problematic systems, many recognize the critical nature and role that LIS plays in many organizations systems. Joiner and Tsai (1998) used the term "legacy software system" and offer this description:

> Legacy software systems are programs that are critical to the operation of companies, but that were developed years ago using early programming languages such as COBOL and FORTRAN. These programs have been maintained for many years by hundreds of programmers, and while many changes have been

> made to the software, the supporting documentation may not be current. These factors contribute to the staggering cost of maintaining legacy systems. (p. 185)

Along the same line, Paradauskas, and Laurikaitis (2006) noted that LIS "contain incredible detailed business rules and form the backbone of the information flow of organization, but their maintenance is very expensive and it is very difficult, if not impossible, to expand them" (p.221). Others combined both views; they recognize the long term investment that LIS plays in organizations, yet they realized that these systems are slow and riddled with problems. Bisbal et al. (1999) provided the following description of Legacy Information System (LIS):

> A legacy information system represents a massive, long-term business investment. Unfortunately, such systems are often brittle, slow, and non-extensible. Capturing legacy system data in a way that can support organizations into the future is an important but relatively new research area. (p. 103)

Reddy and Reddy (2002) explained that legacy systems are often a burden to large and mature companies, and they hamper companies when they face competitive international competition. They explained that these legacy systems are designed to handle particular functional areas but they lack flexibility in terms of their effective use of information technology.

Information Technology literature also mentions LIS in reference to other aspects of technology such as the use of programming language and paradigms. Verbaan (2010), for example, noted that legacy information systems are typically based on older programming languages and paradigms.

In regards to our views and understanding of the term "legacy", we acknowledge that it is used within various contexts. However, we agree mostly with the definition listed on the Wikipedia web site for legacy system, which states that *legacy* is "an old method, technology, and computer system or application program." This definition in our opinion lists the general characteristics of the term without indulging in the underlying difficulties and problems with it.

Legacy Programming Languages and Tools

Applying the meaning of the terms "legacy" and "legacy system" to programming languages, one may conclude that older programming languages are all called "legacy". This is partially true but not all older unused programming languages are called legacy. Take Pascal programming language for example; it is considered to be a thing of the past, but it is rarely cited as legacy. One study, for example, specifically identified COBOL and FORTRAN as two legacy programming languages (Joiner & Tsai, 1998), but did not mention Pascal. Guthery (1993) in another instance lists a number of programming languages that were taught in the 1970s including ALGOL, APLAPT, BASIC, COBOL, FORTRAN, GPSS, JOSS, JOVIAL, LISP, PL/I, SIMULA, and SNOBOL. Most of these languages are no longer in use, yet they are not called legacy.

While some programming languages are described as "legacy" in some literature, COBOL is foremost mentioned as a legacy programming language and the two words, legacy and COBOL, are strongly associated with each other (Haney, 2003). There are two reasons that explain this strong association; first, COBOL is often associated with applications developed for legacy information systems such as are commonly used in banking, insurance, payroll, and other business applications. Second, COBOL is strongly associated with other technologies that are in turn deemed legacy (like batch processing).

COBOL is considered the *backbone* of most legacy information systems. In fact, because of this *backbone* consideration COBOL was mainly to blame for the Y2K and the catastrophic effects that were predicted at that time (Hardgrave & Doke, 2000; McGirr, Barker, Decker, & Schmidt 2004). This kind of association started early in 1980s when COBOL was the most widely used

language for business applications. Banking, retail, insurance, manufacturing, and others all were using COBOL for the development of new systems. Once these systems were developed, they needed to be maintained and COBOL was the preferred programming language for maintenance because COBOL programs were considered to be easier to maintain. This continued until the proliferation of the Internet and e-Commerce. COBOL was not suitable for providing the interface and the front-end functionality that the new e-Commerce applications required. Instead, COBOL worked in most cases to perform the back-end of the process while the front-end and other phases of the e-Commerce phases were developed using other technologies.

The predicament described applied not only to COBOL but also to many tools that were relegated later to back-end services. A number of technologies (tools) that worked with COBOL in the past, continue to work with COBOL. This combination of tools "working with COBOL" formed the foundation of the mainframe technology. The American Programmer web site listed a few of those that worked with COBOL, including REXX, CICS, DB2, SQL, JCL, TSO, ISPF, Assembler, CLIST, Focus, Syncrost, and SFSORT.

Explaining it further, COBOL is called "legacy" not only because it is used in older applications but also because it uses or interfaces technologies that are all deemed legacy. This association of COBOL with "Legacy" has been carried to schools where COBOL is taught, so many students do not see a future for COBOL and would prefer other programming courses that can be used for future applications (Phillips, Ryan, Harden, Guynes, & Windsor, 2013).

COBOL is termed "legacy" more than other programming languages for two reasons; first COBOL is the only old language that is still used today in the industry at a level of significance. Second, COBOL is still used with other technologies (such as CICS, DB2 and others) that are considered legacy.

Selecting a Programming Language to Teach - Then and Now

There are many factors to be taken into consideration when selecting a programming language to teach (and hence replacing one programming language with another). Some of these factors are related to the rapid changes in the IT field and the need to integrate newer technology into the curriculum. Other factors are considered as well. But to appreciate what influences the selection of one programming language over others, some understanding of the history of programming languages is helpful.

This section first introduces a brief history of programming languages and their evolution in academia. The section explains the selection of programming languages in curricula during the 1980s and 1990s, the dates of increasing use of COBOL. It then explains the selection of programming language of recent time since the proliferation of the Internet, GUI, and e-Commerce applications. Our purpose for this sequenced description is to illustrate that selecting a programming language to teach (or deleting a language for this matter) is dependent on a number of factors – some are related to the history of the programming in general while others are related to the rapid advances of the technology.

Programming Languages – A Brief History

Markstrum (2010) explained that the inception of high level programming languages dates back to the 1950s. This coincided with what is described by Duckett (2013) as the heritage of COBOL. Sammet (1972) noted that programming languages started in 1952 "with short code for UNIVAC" (p. 601). Although the inception of programming languages goes back decades, the most

notable change is the substantial increase in the number of programming languages that are in use today.

Kaplan (2010) explained that there are approximately 2,000 to 3,000 well known programming languages. Bergin (2007) noted that there are more than 8,500 programming languages in use at the time of publishing his study. Markstrum (2010) also commented on the large number of programming languages in use but did not specify a number. It is safe to assume that since 2010 the number of programming languages has continued to grow and it exceeds the 8,500 languages mentioned by Bergin. While only a fraction of these may be fully adopted for use in industry, it is far beyond those used when COBOL was first introduced.

Despite the numerous programming languages used today, a smaller number are popular and the use of others remains sporadic. Researchers believe the History of Programming Languages (HOPL) conferences demonstrate the number of programming languages popular in academia (Bergin, 2007; Bergin & Gibson, 1996). The HOPL conference is held for teachers and professors to present the newest programming languages being used in academia. Three HOPL conferences have taken place since 1978. Table 1 below shows the date that HOPL conferences took place and the number of programming languages that were popular at the time of the conference. (Bergin, 2007; Bergin & Gibson, 1996; Wexelblat, 1978).

Table 1 – History of Programming Languages Conferences

Conference	Conference Year	Conference Location	Number & Name of New Programming Languages Presented
HOPL1	June 1-3 1978	Los Angeles, CA	13 – **ALGOL, APLAPT, BASIC, COBOL, FORTRAN, GPSS, JOSS, JOVIAL, LISP, PL/I, SIMULA, SNOBOL**
HOPLII	April 20-23, 1993	Cambridge, MA	14 – **Ada, Algol 68, C, C++, CLU, Discrete Event Simulation Languages, FORMAC, Forth, Icon, Lisp, Monitors and concurrent, Pascal, Prolog, Smalltalk**
HOPLIII	June 9-10, 2007	San Diego, CA	12 – **AppleScript, BETA, C++, Emerald, Erlang, Haskell, High Performance Fortran (HPF), Lua, Modula-2 / Oberon, Self, Statecharts, ZPL**

Although the programming languages that are popular in academia are limited in number (the table above shows 38 languages) they still represent a significant number of programming languages that provide special functions being used at some academic institutions (Henriksen & Kölling, 2004; Utting, Cooper, Kölling, Maloney, & Resnick, 2010). These special function languages are helpful in many instances and they add factors for selection when considering selecting a programming language to teach.

Selecting a Programming Language - Then

By “Then” we are referring to the period in the 1980s and 1990s when teaching COBOL became more popular, with more programs offering courses in COBOL in anticipation of Y2K. At that time, selecting a programming language to teach for a specific course was a simpler task. There were a fewer programming languages to select from and each specialized in one field more than

the others. COBOL was the language for business applications. For example, COBOL handles file processing of financial data and business reporting with accuracy, ease, and efficiency. On the other hand, FORTRAN was the language for scientific applications. In this use, a small amount of error was an acceptable tradeoff to accommodate very large or very small numbers and the ability to use mathematical functions such as sin or cos. FORTRAN provides constructs to rapidly "crunch" scientific data. Lastly, the C programming language was gaining ground as a language for systems programming. With the Unix operating system gaining acceptance, the C language used to implement Unix became a language of study in academia. The strengths of C enabled direct management of computing resources and offered expressive power to manipulate bytes, the foundational building blocks of computer systems. Since these were the dominant languages back "Then", they were the obvious choice to be included in a curriculum. An exception to industry use was BASIC, a language specifically designed for educational purposes to enable the beginner to grasp programming basics before advancing to other languages. BASIC was often used in a first programming course. It is interesting to note that BASIC subsequently gained some popularity in industry and was the genesis for Visual Basic, which due to the market influence of Mircrosoft became, and is still, a significant programming language. Similarly, Pascal was another programming language used for educational purposes that gained acceptance in industry, although the time for Pascal has since past.

COBOL, FORTRAN, C, and BASIC (or Pascal) are considered to be third generation languages as these are targeted at human development and understanding with less concern for the actual operation of a computer. However, an understanding of computer architecture and internal operation is considered to be essential to fully develop programming skills. Thus, the teaching of an assembly language (second generation languages) was considered necessary and continues to be included in many IT programming curriculum (Bolanakis, Evangelakis, Glavas, & Kotsis, 2011).

Criticisms for these programming languages were pointed and clear. COBOL was criticized for its wordiness, overall structure, and the length of program that was needed to complete simple task. BASIC was criticized for over simplicity and for including line numbers that in turn encouraged using the GOTO statement, a nemesis leading to unstructured programming. The "unstructured" result made these programs difficult to follow and maintain. FORTRAN was criticized for brevity and using acronyms instead of full words to refer to commands; this made it difficult for novice programmers to learn the language and make use of the functions the language provides. While C provided expressive power for systems programming, it was perhaps too low level and thus not suited for other applications.

Life used to be simpler for educators to select a programming language to teach in a given course: a small set of programming languages to choose from, clear application and specialized areas, distinct advantages for each language, and specific difficulties with each language. To summarize, the main points of selecting a programming language then (in the 1980s and 1990s) are:

- Small set of significant programming languages.
- Programming paradigms were limited in number and sophistication. There was always talk about sequential, structured, or procedural approaches.
- To address difficulties in learning to program, a few educational languages (BASIC and Pascal) were developed and used. While intended for educational purposes, these were adopted as full fledge programming languages by industry.
- There was clear distinct purpose for each programming language, COBOL for business courses, FORTRAN for scientific application, C for systems programming, and BASIC for beginners.

Selecting a Programming Language – Now

By "Now" we mean the period after Y2K, through the dot com bust, and then through the subsequent proliferation of Internet and related e-commerce applications. With the proliferation of technology came an increasing the number of programming languages, adding to the applications that programming languages deal with and the introduction of new methodologies, systems, classifications and many others (Welton, 2005). This also created many other libraries that can be shared across programming languages. If a programming language is weak in a certain application (like a graphical user interface for a business application), a library can be incorporated to handle and overcome the weakness. These libraries effectively form variations within the programming languages.

In terms of number of programming languages, this number has grown so much that it is hard to keep an accurate count of the languages. Although a limited number of programming languages (like Java, C++, Visual Basic, and others) remained widely used in academia, a significant number of languages are developed and taught by different college programs.

As the number of programming languages increased, so too did the number of programming paradigms. The old list, which was basically procedural and structured, now included event driven, declarative, object oriented, functional, and many more related to the web. As a result, academia is faced with the challenge of selecting a set of languages and paradigms that can fit into a program and that provides the best benefit to its target population (Welton, 2005).

A new factor added to the dilemma is the beginner level programming languages, such as Alice from Carnegie Mellon University. These are designed for one purpose only, to making learning to program easier and, therefore, are purely educational. Unlike BASIC and Pascal, these languages will never be used outside of education. They provide no value for a Computer Science student to claim on their resume. By selecting one of these beginner level languages, the program will take away the potential to teach a language of value. These languages provide significant educational values, so when deciding what language to use, consideration is given to these languages thus complicating the decision of which language to teach (Henriksen & Kölling, 2004; Utting et al., 2010).

Factors for Keeping COBOL

Dunn and Legerfelt (2005) suggested that COBOL cannot be written off as a dead language. Kizior, Carr, and Helpren (2005) noted that there were between 150 and 200 billion lines of COBOL code in business applications at the time of publishing their study and that several billions of lines of COBOL code were added annually to these applications. The implication here is the need to continue teaching COBOL so that graduates with COBOL knowledge are able to handle this massive amount of COBOL code.

COBOL continues to be resilient. Many wrote it off years ago, yet is still taught in academia and used in industry. Thus, in this section, we discuss the factors in favor of continuing to teach COBOL at academic institutions. Although there may be claims for other arguments for keeping COBOL, we will limit these to the common reasons that are cited in programs that still teach COBOL. These include; the supply and demand for COBOL programmers, the best tools for the job, and the association of COBOL with the mainframe.

Supply and Demand

As the number of programs that teach COBOL decline, so to does the number of graduates who possess knowledge of this programming language (Roggio et al., 2013). This creates more demand from the industry for graduates of the *few* programs that continue to teach it. The demand

for COBOL is not growing like the demand for other programming languages but the demand for mainframe jobs is still strong (McGirr et al., 2004). New applications for the mainframe continue to be developed and thus the demand for COBOL continues to grow. Donahue and Power (2012), for example, searched job advertisements at different journals and reported that the demand for legacy job titles is still significant.

Aside from the demand for mainframe jobs, COBOL seems to respond well to the new jobs that are created as a result of advances in technology. VanLengen and Haney (2009) explained that COBOL can be used for creating web services, offsetting notions that COBOL is used mainly for older technology. Duckett (2013) noted that COBOL began to be used for cloud applications, thus opening the door for a wide range of legacy applications to be moved to the cloud.

The increasing demand for COBOL graduates coupled with the shrinking number of schools who graduate students with COBOL knowledge create a shortage at a time of overall high demand. This creates opportunities for graduates with COBOL knowledge to land higher paying jobs.

Best Tool for the Job

In the study entitled "The Economics of Programming Languages", Welton (2005) explained that each programming language is more suitable to one specific task than others. We used the terminology to "best tool for the job" to describe something similar. Take for example a screwdriver; a Phillips screwdriver turns Phillips screws (those with crossed slots) a lot easier than other screw drivers. Therefore, a Phillips screwdriver is the best tool for this kind of application (Phillips screws).

Developing programs using the "Structured" paradigm is best illustrated in COBOL. The layout of COBOL programs into four divisions and the simplicity with which modules can be created make it the best candidate to teach structured programming methodology (Stern, Stern, & Stern, 1999). Added to this, the simplicity with which COBOL programs can be maintained makes it easier to use for business applications. Some even suggested that this simplicity of maintaining COBOL programs helped to avert the catastrophic effects that were predicted from the Y2K (Kizior et al., 2005).

Although COBOL programs are considered longer than programs written in other programming languages, there are factors that make COBOL programs more understandable than others. First, COBOL is very much like the English language. The verbs that are used in COBOL and the division of paragraphs, sections, and statements all resemble the written English language. The absence of confusing symbols (like the brackets, the semi colon, and the two equal signs) makes COBOL less confusing than other languages. In addition, the looping mechanism in COBOL is easier to understand than others.

Even with the newer paradigms that have been developed, COBOL can be adapted. For example, object oriented COBOL is available (Hardgrave & Doke, 2000). COBOL also has proven that it can be used with the newest technology. Hardgrave and Doke give "cloud computing" as one example of the new technology that COBOL can be used with, demonstrating its ability to fit recent technological developments.

The Association with the Mainframe

Many consider COBOL's best use to be applications developed or maintained on the mainframe (Arranga, 2000; Carr & Kizior, 2000). Although this may be perceived as detrimental as it limits the use of COBOL to one type of "antiqued" application (Phillips et al., 2013), the number of applications developed on the mainframe is large and continues to grow.

Paulson (2001) points out that the mainframe is still running critical applications within various companies. Arranga (2000) explained that most companies have many mainframe applications installed. These applications cannot be replaced because the cost to change them is prohibitive to many. The style of working on the mainframe has forced many companies to stay with this old technology simply because it is too expensive to switch to other technologies.

Factors against Keeping COBOL

KVSN and Choudhari (2013) used an analogy when comparing COBOL to other modern programming languages by asking the question "can the elephant run with the deer?" Dunn and Legerfelt (2005) suggested that COBOL is too wordy and out-of-date. Paulson (2001) expected that COBOL "is going out the way of Latin, obsolete save for some critical text" (p. 13).

Phillips et al. (2013) explained that most students are not motivated to learn COBOL. The same authors attempted to motivate the students to learn COBOL and acquire the mainframe skills. They enrolled their students in the IBM Master of the Mainframe Contest. Yet despite their effort, they noted that, in the start, out of the over 200 colleges and universities in Texas, only 8 participated in the IBM academic initiative to teach mainframe skills. This short list of colleges (8 out of 200) indicates that motivating students to learn COBOL is a hard sale.

The above disadvantages are a few that are whispered loudly in many academic institutions regarding the teaching of COBOL. To add clarity to the reasons for not selecting COBOL, we need to be more specific than the general reasons given above. In the remainder of this section we discuss the factors against teaching COBOL. Although this list is not comprehensive, it conveys the substance of the argument against teaching COBOL.

The Likability (or Lack of It) Factor for COBOL

Baldwin and Kuljis (2001) noted that the majority of students (even students enrolled in CS programs) find learning to program a difficult task. In the same study, Baldwin and Kuljis observed that students perceive programming as an "unnecessary evil". For COBOL the issue is even worse as it is the most disliked by students (Paulson, 2001). Some of the reasons that contribute to the unpopularity of COBOL are related to the general difficulty in learning to programs. But others are related to COBOL program itself.

Educators in the field of information technology have accepted that learning to program computer languages is considered a difficult task for most of students for three main reasons: the rigid syntax, the unfamiliar structure, and the insignificant output generated from long time working on a program (Ali & Smith, 2014; Kelleher & Pausch, 2005). COBOL fully embodies all these factors.

The syntax in COBOL is rigid like all other programming languages. However, the division of the COBOL programs makes tracking syntactical error messages more difficult. An error showing in one division may be caused by inconsistency in other divisions. At the same time, the structure in COBOL is unfamiliar to many. This creates a difficult in comparing the structure of COBOL programs to common popular events in life, thus creating analogy is difficult in this case.

Perhaps the most difficult aspect of COBOL programs is the lack of interesting applications. A typical application of COBOL programs that are shown at programming courses may include batch applications, line counting, simple data retrieval, report creation, and others. This kind of applications is far from the visual applications that make programming more interesting and provides for more relevant practical applications (Baldwin & Kuljis, 2001).

No New Development – No Future

Practitioners in the IT field classify application development into two general categories: developing new applications (new development) and maintaining old programs (maintenance). While there is an abundance of maintenance on old COBOL programs, few new applications have been developed with COBOL and little (if any) new development with COBOL is expected in the future. Thus, it is expected that future COBOL programmers will continue on the current set or applications.

Advocates of COBOL (Arranga, 2000; Carr & Kizior, 2000; Kizior et al., 2005) note that there are billions of lines of COBOL code that need to be maintained and this staggering number means continuous jobs for COBOL programmers. However, the fact is that there are a **finite** number of these lines of code that need to be maintained. Given that there is no new development in COBOL (companies do not use COBOL to develop new applications), this finite number may end in the future and potentially it may mean there will be no future for COBOL programming jobs.

Dwindling Market Base

McLean (2006) noted that hard skills (such as learning programming languages) get the students a "foot in the door" when it comes to finding jobs in the industry. Another study noted that when employers search resumes, they look for keywords (such as name of programming languages) to select potential candidates (Koong, Liu, & Liu, 2002). So to better serve students, it is logical to teach them skills that they can place on their resume.

Although there is still demand for COBOL, the demand is limited to mainframe jobs. Even with these mainframe jobs, some companies are moving away from COBOL (Paulson, 2001). In other words, the market base for COBOL is limited to one area: the mainframe. This is contrast to other programming languages (like Java, Visual Basic, C++) where the market base is not limited.

Surendra and Denton (2009) stressed the importance of designing more relevance into future IS curricula. Relevance of a programming language increases when there is wider range of applications. Limiting the market base to smaller number of applications is liable to decrease this relevance and then the interest of students to take such course may be compromised. Roggio, Comer, and Brauda (2003) noted that academic programs teaching COBOL are steadily decreasing while courses in Java and Visual Basic are on the rise. This shows the limitations of having dwindling market base for COBOL jobs.

Arguments and Recent Development

The authors are two faculty members with opposing views on whether to keep teaching COBOL or to replace it with another programming language. The first faculty member (Faculty #1) teaches at the department of Information System and Decision Sciences (ISDS) at IUP. Although this department does not teach COBOL Faculty#1 taught COBOL for many years and has extensive industry experience working with COBOL. The second faculty (Faculty #2) teaches at the computer science department and is involved in teaching COBOL at the department. Furthermore Faculty #2 has worked with the mainframe and COBOL for many of years. Each has his own argument regarding teaching COBOL, thus we present both arguments below.

Faculty #1 Arguments

COBOL is a liability to any program that teaches it. Students do not like taking COBOL and it gives bad name to the program. There is a limited number of programming courses that students can take during their time at college. Let the old COBOL not be one of them. Students want to

add keywords to their resume to represent the skills that will help them to enter the job market. Skills in COBOL, CICS, JCL, or similar are not marketable to the industry at large.

Enrollment in IT programs is dropping mainly due to the difficulty associated with learning to program. Students (even some enrolled at CS programs) find learning to program a difficult task mainly for two reasons: first the unfamiliar structure in computer programming, and second the uninteresting nature of applications that some programming courses have students complete. COBOL has both these features. First the structure of COBOL (the four divisions for example) is difficult for the students to learn. Second, most of the applications in COBOL involve developing reports, line counting, batch processing, and many other similar applications that students find uninteresting at the time of GUI, mobile and cloud computing.

The existing programming course can be replaced to use Visual Basic.NET. With Visual Basic (VB), students can learn about a newer paradigm called "event driven programming". Many of the applications are visual and thus appeal to students as well as the public. The interface of VB with other applications and with the web gives it enormous advantage. Simply put, it is easier to interface programs written in VB with other applications than with any other programming languages. Furthermore, VB is simpler to learn, enables applications to be developed quickly, and provides a friendly development environment. Also, the newer version allows VB to be taught using the old procedural methods together with the newer object-oriented methods.

As IT educators, we are all aware of the decline of enrollment in our programs nationally and internationally. All blame is directed at programming courses. Students do not like to take programming courses because programming is difficult to learn and they spend so much time writing a program to produce some simple output they call "stupid". Programs that do line counting, sequential file process and simple output with the display/accept statements are among those simple output programs. With COBOL, the faculty is limited to teaching these kinds of applications that I mentioned.

We (IT educators) are responsible for dealing with this enrollment decline in our majors. We have to make our programming courses more interesting to the students. You cannot make writing COBOL programs interesting but you can with Visual Basic. With the GUI applications and the simplicity of the interfaces that Visual Basic provides, interesting programs can be developed. Programs which interface with the web or with other applications (like Word, Excel, or Access) are indeed interesting applications that can be developed. With Visual Basic, you can write a program that interfaces with social media, for example, to make interesting programs that far exceed the benefits gained from line counting and sequential file access programs typically written in COBOL.

Faculty #2 Arguments

There are billions of lines of code that need to be maintained and the dwindling number of graduates trained in COBOL is quickly disappearing. The market demand is strong and will continue to be strong for some time. As the coordinator for the department's internship program I interact with many companies. About fifteen percent of the internship companies use a mainframe. But this fifteen percent are major corporations and thus twenty to twenty five percent of the intern positions are mainframe related. Furthermore, from surveys and reports of our recent graduates, the salaries of COBOL and mainframe jobs are typically higher than others. I am convinced that COBOL on the mainframe will continue to be around for many years to come.

COBOL and the mainframe provide exposure to a different operating system and development environment from what they have grown up with and used in college. Yes, this difference may be received with dislike and it certainly has a "legacy" feel. However, it is an enterprise level system servicing high volumes of throughput and concurrent users. Exposure to the mainframe

enables a balanced understanding of today's computing. While a large majority of students will not pursue a career in COBOL programming on the mainframe, this exposure will be of benefit wherever their career takes them. I started my career in COBOL/mainframe, moved to C/Unix on mini-computers, then Java on PCs before moving into academia. I found my COBOL/mainframe experience to be of value throughout my career.

To address the "legacy" feel I would advocate using newer mainframe development tools such as RD/Z. RD/Z is basically the Eclipse development environment retarget for COBOL on the mainframe. Eclipse is an environment for Java with which many students already gain familiarity in other courses. Let's leverage this in introducing students to mainframe computing. After attaining an appreciation for the power of the mainframe, more traditional mainframe tools can be introduced.

I will concede that COBOL would no longer be active had it not been for the endurance of mainframe systems. It is interesting to note that minicomputer systems, such as DEC, have been squeezed out between the mainframe and Intel based servers. But the IBM mainframe remains strong, and it is still the computing backbone of major corporations, especially finance and related. All direct competitors have since past.

Recent Development

A recent development has taken place with the authors of this paper that added a twist to the debate about COBOL and affected the conclusion of our study. Both authors recently attended a conference sponsored by IBM that calls for the return of teaching COBOL and the mainframe into the curriculum. The conference was titled "2014 IBM Academic Initiative Enterprise Systems Educators Conference" and was geared toward promoting the new "system Z" that uses COBOL to access Mainframe applications. The basic points of the workshop can be summarized in the following points:

- COBOL is tied to the mainframe and the mainframe is going to be around for many years.
- There are billions lines of COBOL code that need to be maintained, experts in COBOL and the mainframe are needed to keep these systems up and running.
- The workforce that is working on the mainframe is aging and most are close to retirement. Therefore new recruits with solid COBOL and mainframe skills are needed to fill this vacancy.
- Due to the shortage of new graduates working trained on COBOL, the demand and salary for these jobs is increasing.
- One person said it best, "we need graduates who are trained in COBOL."
- Thus it will benefit CS programs as well as the industry if academia returns to teaching COBOL in their programs.

The points expressed in the workshop did not totally change the mind of Faculty #1, but it did persuade him that it may be useful for the time being to keep teaching COBOL. However, concerns over the weaknesses, difficulty, and inappropriateness of COBOL to CS students remain the same. Furthermore, there need to be steps taken to minimize the negative effect of COBOL. These points need to be addressed when submitting recommendations about the teaching of this course.

Conclusion/Recommendation

The objective that we stated at the beginning of this paper is to come to a conclusion regarding the teaching of our COSC Application Programming course. The debate that we presented in this paper has taught us a number of lessons. We learned from the literature review the difficulty as-

sociated with replacing COBOL with another language. We also discovered the wide range of modern languages that are available to teach. Some of these languages are quick to appear and yet quick to disappear from curriculum as well. From attending the "2014 IBM Academic Initiative Enterprise Systems Educators Conference" we learnt the directions that some programs are taking regarding the teaching of COBOL and their justifications for their directions.

Based on the literature review conducted for this paper, the ongoing discussion between the two authors, and the findings from the IBM workshop that the authors attended, this study submits the following recommendations regarding the teaching of COBOL in the COSC 220 Application Programming course:

- Keep teaching one course (4 credits) in COBOL.
- Change the name of the course to a more marketable name such as enterprise application development, enterprise computer programs (2014 IBM Academic Initiative Enterprise Systems Educators Conference).
- Seek partnership with other companies that need graduates trained in COBOL.
- Change the focus of the course to incorporate some visual application to make the course more interesting.
- Consider using a modern development environment such as RD/Z and incorporate more interesting and appealing applications. For example, explore new interfaces with the web and cloud computing.

As one of our stated goals at the beginning of this paper, we attempted is this study to reach a consensus for teaching COBOL at the COSC department at IUP. Through our literature review and through our long debate and also based on the information that we gathered at the "2014 IBM Academic Initiative Enterprise Systems Educators Conference" we have reached a conclusion that it will be best for the COSC program at IUP to continue teaching COBOL in the COSC 220 course.

References

Ali, A., & Smith, D. (2014). Teaching an introductory programming language in a general education course. *Journal of Information Technology Education: Innovations in Practice, 13*(6), 57-67. Retrieved from http://www.jite.org/documents/Vol13/JITEv13IIPp057-067Ali0496.pdf

Arranga, E. (2000). Roundtable Discussion, "In COBOL's defense." *IEEE Software, 17*(2), 70-72.

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/

Bergin, T. J. T. (2007). A history of the history of programming languages. *Communications of the ACM*, *50*(5), 69-74.

Bergin, T. J., & Gibson, R. G. (Eds.). (1996). *History of programming languages II*. New York: ACM Press.

Bisbal, J., Lawless, D., Wu, B., & Grimson, J. (1999). Legacy information systems: Issues and directions. *IEEE software*, *16*(5), 103-111.

Bolanakis, D., Evangelakis, G., Glavas, E., & Kotsis, K. (2011). A teaching approach for bridging the gap between low-level and high-level programming using assembly language learning for small microcontrollers. *Computer Applications in Engineering Education, 19*(3), 525-537.

Carr, D., & Kizior, R. J. (2000). The case for continued COBOL education. *Software, IEEE*, *17*(2), 33-36.

Comella-Dorda, S., Wallnau, K., Seacord, R. C., & Robert, J. (2000). *A survey of legacy system modernization approaches* (No. CMU/SEI-2000-TN-003). Carnegie-Mellon University, Pittsburgh, PA, Software Engineering Inst.

Donohue, P., & Power, N. (2012, May). Legacy job titles in IT: the search for clarity. *Proceedings of the 50th Annual Conference on Computers and People Research* (pp. 5-10). ACM

Duckett, C. (2013, November 21,). *COBOL still not dead yet, taking on the cloud.* ZDNet. Retrieved June 25, 2014 from http://www.zdnet.com/cobol-still-not-dead-yet-taking-on-the-cloud-7000023462/

Dunn, D. L., & Lingerfelt, D. F. (2005). Can Visual Basic replace COBOL? … and should it? *Journal of Computing Sciences in Colleges, 20*(4), 6-12.

Gabriel, R. P., & Steele Jr, G. L. (2008, October). A pattern of language evolution. In *Celebrating the 50th Anniversary of Lisp* (p. 1). ACM.

Guthery, S. (1993). Are we still having fun? A minority report from HOPL-II. *ACM SIGPLAN Notices, 28*(8), 1.

Haney, J. D. (2003). Something lost - something gained: from COBOL to JAVA to C# in intermediate programming courses. *Journal of Computing Sciences in Colleges , 22*(5), 227-234.

Hardgrave, B. C., & Doke, E. R. (2000). COBOL in an object-oriented world: A learning perspective. *IEEE Software, 17*(2), 26-29.

Henriksen, P., & Kölling, M. (2004, October). Greenfoot: Combining object visualisation with interaction. In *Companion to the 19th annual ACM SIGPLAN conference on Object-oriented programming systems, languages, and applications* (pp. 73-82). ACM.

HFC Computer Information System. (2014). *Is there still a market for Cobol skills/developers?* Henry Ford College. Retrieved June 25, 2014 from http://cis.hfcc.edu/faq/cobol

Hurwitz, M. (2011). The impact of legacy status on undergraduate admissions at elite colleges and universities. *Economics of Education Review, 30*(3), 480-492.

Joiner, J. K., & Tsai, W. T. (1998). Re-engineering legacy COBOL programs. *Communications of the ACM, 41*(5es), 185-197.

Kaplan, R. M. (2010, October). Choosing a first programming language. In *Proceedings of the 2010 ACM conference on Information technology education* (pp. 163-164). ACM.

Kelleher, C., & Pausch, R. (2005). Lowering the barriers to programming: A taxonomy of programming environments and languages for novice programmers. *ACM Computing Surveys (CSUR), 37*(2), 83-137.

Kizior, R. J., Carr, D., & Halpern, P. (2005). Does COBOL Have a Future? In *The Proceedings of the Information Systems Education Conference 2000* (Vol. 17, p. 126).

Koong, K. S., Liu, L. C., & Liu, X. (2002). A study of the demand for information technology professionals in selected internet job portals. *Journal of Information Systems Education, 13*(1), 21-28.

KVSN, S., & Choudhari, A. (2013). Legacy mainframe back-ends supporting new age enterprise applications: Can the elephant run with deers? *Proceedings of the 6th India Software Engineering Conference* (ISEC '13). ACM, New York, NY, USA, 55-60..

Laguna, M. A., & Crespo, Y. (2013). A systematic mapping study on software product line evolution: From legacy system reengineering to product line refactoring. *Science of Computer Programming, 78*(8), 1010-1034.

Legacy. (2014). In *Merriam-Webster.com.* Retrieved June 20, 2014 from http://www.merriam-webster.com/dictionary/legacy

Markstrum, S. (2010, October). Staking claims: A history of programming language design claims and evidence: a positional work in progress. In *Evaluation and usability of programming languages and tools* (p. 7). ACM.

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

McGirr, G., Barker, R., Decker, J., & Schmidt, G. (2004). Crisis with COBOL: Where industry and IT *Departments Sta*nd. *Proceedings of the Midwest Instruction and Computing Symposium*, University of Minnesota, Morris, April 16-17, 2004

Paulson, L. D. (2001). Mainframes, COBOL still popular. *IT Professional*, *3*(5), 12-14.

Paradauskas, B., & Laurikaitis, A. (2006). Business knowledge extraction from legacy information systems. *Rn, 2*(1), R2.

Phillips, B. K., Ryan, S., Harden, G., Guynes, C. S., & Windsor, J. (2013, May). Motivating students to acquire mainframe skills. In *Proceedings of the 2013 Annual Conference on Computers and People Research* (pp. 73-78). ACM.

Reddy, S. B., & Reddy, R. (2002). Competitive agility and the challenge of legacy information systems. *Industrial Management & Data Systems, 102*(1), 5-16.

Roggio, R. F., Comer, J. R., & Brauda, P. (2003). IS programs become accredited: COBOL in crisis. *Information Systems Education Journal*, *1*(15), 1-10.

Sammet, J. E. (1972). Programming languages: History and future. *Communications of the ACM*, *15*(7), 601-610.

Stern, N. B., Stern, R. A., & Stern, N. (1999). *Structured COBOL programming*. Wiley.

Surendra, N. C., & Denton, J. W. (2009). Designing IS curricula for practical relevance: Applying baseball's" Moneyball" theory. *Journal of Information Systems Education*, *20*(1), 77-86.

Utting, I., Cooper, S., Kölling, M., Maloney, J., & Resnick, M. (2010). Alice, Greenfoot, and Scratch--A discussion. *ACM Transactions on Computing Education (TOCE), 10*(4), 17.

Verbaan, M. (2010). Legacy information systems, Can they be agile? A framework for assessing agility. (English). *Communications of the IIMA, 10*(4), 1-14.

VanLengen, C. A., & Haney, J. D. (2009). Creating web services for legacy COBOL. *Information Systems Education Journal, 7*(28), 1-10.

Welton, D. (2005). *The economics of programming languages*. Retrieved March, 26, 2014 from http://www.welton.it/articles/programming_language_economics.html

Wexelblat, R. L. (1978). *History of programming languages I*. ACM.

Whitson, G. (2007). From advanced COBOL to data, file and object structures. *Journal of Computing Sciences in Colleges, 22*(5), 39-45.

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 of 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.