The Impact of New Programming Languages on University Curriculum

Katie L. Emigh
Computer Applications Department, Grand Rapids Community College
151 Fountain NE, Grand Rapids, MI 49503

Abstract

This paper covers the impact that the seemingly continuous introduction of new programming languages has on the computer information systems curriculum in a university. Computer information systems departments are currently implementing changes based on the newest languages, Java and C#. The relationship between universities and the corporations behind these languages, greatly affects how companies interact with these institutions. Before implementing the latest languages, universities must address if they should continue to offer traditional languages such as COBOL. This report provides conclusions and recommendations on the transformation a curriculum must undergo to maintain high levels of enrollment and also demonstrates why it is a challenge for universities to keep an up-to-date language program.

Keywords: Programming languages, computer information systems, COBOL, object oriented languages, curriculum

1. INTRODUCTION

Implementing changes to a programming language curriculum in colleges and universities has been, and will likely always be, a divisive topic in computer information systems departments. Aside from normal curriculum issues, the constant introduction of new programming languages present unique considerations departments must concern themselves with. A clear relationship between the corporations behind the newest languages, Java and C#, and universities demonstrates the potential dangers with changing the curriculum to include these technologies.

Programming is the core of the computer information systems curriculum. Universities are in the midst of struggling with setting aside traditional programming courses like COBOL and placing their focus on object oriented languages. However, with the many factors affecting curriculum changes, computer information systems departments often find themselves with stalemate programs. In addition to the many issues to consider, departments must also take into account several questions in order to maintain high levels of enrollment and to continue to meet expectations of both students and the information technology industry that often drives their curriculum.

2. THE CURRICULUM CHANGE

A clear relationship exists between universities and software developers. From operating systems to programming languages, a strong relationship has always existed between how a university chooses to accept these new technologies and how a company chooses to implement them. After the original development of UNIX from Bell Labs in the early 1970's, universities began to modify the code to make it work on different machines. Today, as companies develop and plan the implementation of new technologies, they rely on the fact that universities are not only a place for many to learn their innovation, but also an institution for their technology to grow. With this relationship in mind, universities must proceed with caution and consider several factors when changing their computer information systems curriculum.

2.1 Effects on Implementing a New Course

As the field of computer science continues to grow at a rapid pace, implementing new courses each semester as a way to keep the curriculum current is now more necessary than ever before. Keeping up with the latest technology, however, poses many problems for universities. From the ever-present issue of dealing with bureaucratic red tape when working with institutions receiving government aide (and most do), to finding qualified instructors to teach the newest concepts, computer information systems departments often find themselves fighting an uphill battle.

The cost of new technology is still a leading factor when it comes to deciding whether to implement a new course in the curriculum. Computer hardware typically becomes obsolete approximately six years after the

initial purchase, and is replaced at an average cost of $2,500. In order to offset the high costs associated with system maintenance and budgetary constraints, many universities are forced to institute a student technology fee that can often range between $25 and $150 per student per year (McKinney 1996). Universities adding fees to the already high cost of education run the risk of driving the price of tuition out of reach for the majority of students. With this in mind, many institutions are unable to adopt curriculum changes because the requirements of a new programming tool often demands system resources above their current hardware and software.

Much of the computer information systems curriculum is driven by what goes on in the business world. Advisory boards made up of corporate representatives, exist to communicate to universities what they want in a graduate. This would indicate that in addition to budget constraints and other factors related to changing the curriculum, computer information systems departments must also concern themselves with the demands of the industry. Research has shown that the definition of what makes a good graduate differs between the information technology industry and the university faculty. At some universities, COBOL is still the foundation and seen as one of the most critical requirements for a first job, while employers are beginning to look for students with experience in a multitude of programming environments (Mawhinney 1998). Universities must choose whether to stick with their core program that has been the mainstay for decades, or break free and begin moving toward a curriculum that will offer the varied programming experiences so coveted by the industry.

2.2 Keeping the Curriculum Current

Aside from normal curriculum problems such as accreditation concerns, course development plans, budget constraints, and finding qualified instructors, the programming language curriculum faces some unique challenges. Issues such as choosing the best language to teach, keeping up with the pace of new types of languages, and knowing how long a language will be viable, are all problems a programming language curriculum must address.

The broad range of topics students study can easily overwhelm them if a university lacks sufficient resources. In the past, a curriculum had at most two programming languages to teach: COBOL and Fortran. Now, some universities struggle with teaching a variety of languages such as Java, C++, C, Perl, and COBOL. Lacking sufficient funds in the budget, some universities attempt to teach these languages without having proper compilers or hardware to facilitate the learning. Students are forced to struggle with the concepts and the syntax, without the proper environment to test their code (MSDN Magazine 2000).

Research on how the programming language is taught indicates several new issues as more and more programming environments are bogged down with *bells and whistles*. One study shows that since the introduction of recent programming languages such as Java, more research needs to focus on the expanding knowledge of what students and teachers learn, and how that affects their perception of understanding programming concepts. The study of computer programming by Douglas Clements investigates how the unique features of various programming environments interact with the goals and content of the subject matter domains (Clements). It is suggested that the many features of certain environments, such as Microsoft's Visual Studio and Sun's Java Development Kit, may interfere with how a teacher promotes learning and development of core programming concepts.

Developing a programming language curriculum that is up-to-date also requires that a university is willing to invest in the education of its educators. The lack of knowledge of instructors results in no new curriculum development (Kuras 1999). Teachers must continuously enrich their qualifications, implement new training methods and techniques supplemented with practical experience; while teaching a new language that is as new to them as it is to their class. With current heavy course loads, budget constraints, lack of sufficient resources, and little support from outside influences such as advisory boards and government regulators, teachers find themselves forced to take on the additional work with little compensation and no easing of their current class load (Gal-Ezer 1998). This burden is something that many say is just too much – and they continue to teach an outdated curriculum that is of little use to a new graduate.

3. WHAT IS THE RIGHT LANGUAGE?

Programming is the heart of virtually all computer information systems programs. More so than operating systems, a greater variety of choices in programming languages in academics have existed over the past twenty years. Traditionally, languages with substantial academic usage have been Pascal, C, COBOL, and Ada. More recently, C++, Java, and the latest introduction of C# have been making their way into computer information systems curriculum. How can a university know they are choosing the best language to teach?

Bjarne Stroustrup, the creator of C++, states that a programming language must be a multitude of things to serve its diverse set of users. The only thing that a language cannot be to survive is a mere collection of 'neat' features (Stroustrup 1994). As computer information systems departments enjoy the increasing enrollment numbers, they also face the problem of designing programming courses for these new students. Today's students do not seem interested or equipped to handle the rigor of a traditional computer information systems program. Students enter programs wanting to learn specific applications like Visual BASIC and

Visual C++ and their 'neat' features, instead of data structures and algorithms (Koffman 1999).

Some universities are responding to the hordes of students who would rather push aside the knowledge of programming concepts where it might not matter what language to teach, and are focusing on training students on technicalities of a particular language. Yet, no matter which direction the curriculum places its emphasis, the larger dilemma still remains. If a university trains on technicalities, they simply use multiple programming languages. However, if a university opts to teach the core concepts, answering what is the best language to teach becomes; how can the language be used as a tool to teach programming concepts? But even those universities still struggle with what is the right language to teach and how can they successfully modify their program to incorporate the latest technology and continue on with the foundation languages.

3.1 When to Stop Teaching a Language

Moving to new programming languages such as Java and C# and pushing aside older languages is not just a technical issue. One theory on the success of a programming language is that the acceptance of a new language is also a social and evolutionary process (Gabriel 1996). Many students who attend universities to learn the newest programming language are experienced programmers already working in the field. Universities face a new challenge of teaching Java or C# to programmers whose popular image of what a language should be is quite different from the language they already have a close bond with. Yet their employer, or the lack of current job skills, forces them to break ties and learn the latest fad.

Some believe that a student can forget Pascal and Fortran the minute they walk off campus. C usage is only important because C++ and Java borrow heavily from it. And while C++ has only been around for ten years, Java has already outdated it (Swanke 1999). What will C# do? How does a university know when to stop teaching a particular language, if at all?

The transition from procedural to object oriented programming has been a struggle for many universities. While most have stopped teaching Fortran, many universities are having a hard time when it comes to parting with COBOL. For more than fifteen years, programmers and educators have been asking 'Is COBOL dead?', 'I thought COBOL was dead', 'Should we continue to teach COBOL?' (MacDonald 1989). For several decades, COBOL has been the central language of programming curriculum. But at some point there has to be a change. Or not?

Keeping COBOL as a foundation in computer information systems curriculum has several benefits. A young programmer who knows COBOL can enter a new job and quickly establish a way to relate with experienced programmers who saw the beginning of procedural languages. Veteran programmers who may have emotional ties to COBOL may not view the freshman as a threat with his/her knowledge of the newest technologies if they can relate with common COBOL. Even if the new programmer uses their skills in Java more frequently than COBOL, a bridge between first-generation procedural programmers and second-generation object oriented junkies can be established.

COBOL is a mature language that has been an integral part of business applications for over 40 years. There are billions of lines of code currently in use (MacDonald 1989). While there is a vast cadre of experienced programmers, many of them are near retirement. Companies can opt to do mass overhauls of programs, however, they may not have the sufficient financial resources to handle such a change. Furthermore, some legacy systems cannot be 're-written' with the newer object oriented languages. The ease of data file manipulation in COBOL is not present in the newer languages. The simplest solution for many companies would be to hire more COBOL programmers – perhaps a solid business reason to keep COBOL in the curriculum. If Java, C++, or C# cannot completely replace COBOL in the business world, should it replace it in the curriculum?

Additionally, COBOL standards are still supported by ANSI. While that issues is a matter of debate in the COBOL community, the COBOL world is changing (COBOL Report 2001). COBOL is now used to create web applications and object oriented methodologies. The community is currently upgrading their COBOL skills in order to teach the new tools for the next millennium. Many argue that there may not be a need to go to another language for web access, thus having to retrain or replace existing COBOL programmers may not be an issue. But, the question remains; how does the computer information systems educator know?

3.2 Java and C#

Many universities have been teaching Java since its introduction. Some are scurrying to make plans for C#. Others are waiting. Waiting for what?

The Computing Curricula 2001 (CC2001) project, which is due out the summer of 2001, will impact how many universities modify their programming curriculum. The task force responsible for creating CC2001 will focus on defining the body of knowledge associated with several areas including Programming Fundamentals and Programming Languages (Joint Task Force on Computing Curricula 2001). Many universities are waiting for CC2001 to restructure their programming curriculum – perhaps hoping for a suggestion on how to implement the newest languages.

Under new guidelines proposed by the Accreditation Board for Engineering and Technology and the

Computing Sciences Accreditation Board, computer information systems departments will have greater flexibility than the past, but must provide a coherent rationale for their curriculum. Curricula 2001 anticipates a plan to assist computer information systems departments in designing their computing curriculum and prepare them for the rationale required by the new accreditation criteria. CC2001 will address the role of programming and offer suggestions on how computer information systems departments can offer skills and training that meets many of the needs expressed by students, employers, and non-CS faculty, and still meet the needs of universities that want to retain their model that is focused on keeping programs centered on fundamental programming issues (Joint Task for on Computing Curricula 2001).

Microsoft's newest object oriented programming language, C#, was introduced in 2000. C# is an object-oriented programming language designed to exploit the power of XML-based web services on the .NET platform (Microsoft Press Pass 2000), similar to the power of JAVA minus the .NET. As more programming languages continue to be introduced, universities must carefully weigh the reasons behind the new languages existence. Will teaching the newest, hottest language actually improve computer information systems curriculum? Do universities really need to adopt another language just because it exists or are they falling victim to the developers competitive market?

For several years, Microsoft has been in the education market making several offers to universities. In March 1998, Microsoft sealed a $6 million deal with Indiana University to upgrade its infrastructure and to supply students and faculty with Microsoft products including their programming studios. Under a new licensing program, the Microsoft Campus Agreement, the company encourages colleges and universities to offer its products to faculty, staff, and students in discounted packages (Krigel 1998). Is this because Microsoft's leader Bill Gates is a great philanthropist or he sees the investment as security for his company? Over 100,000 students at Indiana University will be taught with Microsoft's applications. Why would they switch when they leave the university for the job market?

Some universities may be weary of Microsoft's motives and offers – putting off implementing C# into the curriculum until it is not only a success in other universities, but also a needed employment qualification in area businesses. In June 1998, California State University (CSU) pulled out of deal similar to the one with Indiana University. Microsoft's plan was strongly opposed by students and staff from CSU's 23 campuses who feared it would limit their choice. Many viewed Microsoft's Campus Agreement as a continuation of the type of anticompetitive behavior that has invited government investigations of the company (Macavinta 1998). With Microsoft implementing this new agreement just prior to the introduction of C#, some might view this as Microsoft's way of pushing C# on curriculum that already offers Sun's Java. With many universities struggling for funds to update technology, they are easy prey to offers like those from Microsoft – somewhat strong-armed into new applications even if it does not fit into their curriculum values.

When a university is faced with updating their curriculum, they must also address the issue of programming life cycles. In a university, it takes at least four years for a student to graduate and start working as a programmer. Even if the current curriculum teaches a new programming language such as Java, there is no guarantee that language will be the language employers look for when the student enters the work force nearly four years after they begin their studies. Many universities who teach C/C++ as their core programming language are looking to migrate to Java. Factoring in the two years it could take to implement the new course and the four years it takes for a student to graduate, a newer programming language could dominate the industry before the students mails out his/her first resume. Calculating the probability of a language's existence is an art – not a science.

4. CONCLUSIONS

The impact of new programming languages on a college curriculum has an effect on the local industry, students, and the university as a whole. Retaining the fundamentals of programming will ensure a successful program. While some languages are better than others, any language can be used for that purpose. What universities must attend to is the demand in the workplace. When deciding how and if new languages should be implemented, consider some of the following questions:

1. Is the curriculum focused on knowledge-based or skill-based?

   If the curriculum focuses on core concepts, then being the first university in the community to teach C# may not be so crucial. The main focus is on educating students with the concepts of data models, algorithmic skills, and other fundamental programming theories. The syntax and capabilities of a particular environment can be learned easily after a student fully comprehends the complete concept.

   If the curriculum focuses on training students in the multiple environments that industry employers are searching for, the curriculum should quickly adopt the newest language once it has been established. Students typically are interested in how to use a specific programming environment.

2. Does the curriculum place emphasis on an objects-first model or a functional-first model?

   If the curriculum follows an objects-first model, then the program should adopt the newest programming language as soon as possible. This type of system emphasizes the principles of object-oriented programming and design in the first courses of a curriculum (Joint Task Force on Computing Curricula 2001). In order for this concept to be a success, students should have exposure to object-oriented environments.

   If the curriculum follows a functional-first model, this provides more time to allow the newest language to succeed before a curriculum must adopt it. Since the courses on object-oriented programming come later in the program, it is not as critical to implement C# as soon as the language hits the market.

3. What are area employers asking for from computer information systems graduates?

   If the greatest requirement in a new graduate from an employer is that they have experience in a multitude of the latest programming languages, the curriculum should be able to train students quickly in the newest environments. Keep in mind the fast growing technical centers whose main focus in on this type of training. The enrollment could drop significantly if a university does not respond to the pressure of industry desires.

   If the curriculum focuses on core programming concepts and turns out a high percentage of students who are able to obtain employment in the industry right out of college, then the impact of the newest programming languages will be less. While the curriculum will still need to respond, the magnitude in which it adopts the new languages is not as great as other institutions.

How a computer information systems department responds to the implementation of a new programming language is crucial to their survival. From the vital issue of concluding why languages exist and guessing how long a language will be viable; to knowing when a language should be removed from the curriculum, are all critical concerns that must be addressed. No matter the speed of implementing new programming languages, the fact that it must happen is inevitable.

5. REFERENCES

Clements, D., "The Future of Educational Computing Research: The Case of Computer Programming". *Information Technology in Childhood Education Annual,* 147-179

COBOL Report, 2001, "COBOL Faculty Grant Program". *The COBOL Report* URL: http://cobolreport.com

COBOL Report, 2001, "Reaffirmation or withdrawal of the ANSI COBOL standard?" *The COBOL Report* URL: http://cobolreport.com

Gabriel, R. P., 1996, Patterns of Software. Oxford University Press

Gal-Ezer, J., and D. Harel, 1998, "What (Else) Should CS Educators Know?" *Communications of the ACM*

Joint Task Force on Computing Curricula, 2001, Computing Curricula 2001. *ACM* Ironman Draft Volume: II

Koffman E., 1999, "IT Programs and CS Departments". *Communications of the ACM,* 417-418

Krigel, B., 1998, "Microsoft's new college curriculum". CNET.com URL: http://www.news.cnet.com/news

Kuras, M., 1999, "Changing IS Curriculum and Methods of Instruction". *Communications of the ACM,* 36-39

Macavinta, C., 1998, "College tech deal folds". CNET.com URL: http://www.news.cnet.com/news

MacDonald, L., 1989, "COBOL: Still the major language for business applications programmers". *Journal of Information Systems*

Mawhinney, C., J. Morrel, G. Morris, and S. Monroe, 1998, "Updating the IS Curriculum: Faculty Perceptions of Industry Needs". *Communications of the ACM,* 219-221

McKinney, K., 1996, "Technology in Community Colleges". *ERIC Digest*

Microsoft Press Pass, 2000, "Microsoft Introduces Highly Productive .NET Programming Language:

C#". Microsoft URL: http://www.microsoft….Pass/press/2000/jun00/CsharpPR.asp

MSDN Magazine, 2000, "Sharp New Language: C# Offers the Power of C++ and Simplicity of Visual Basic". Microsoft URL: http://msdn.microsoft.com/msdnmag/issues/0900/csharp/csharp.asp

Rebelsky, S., 2000, A Web of Resources for Introductory Computer Science

Rocheleau, B., 1997, "Computing in Higher Educational Institutions in an Era of Sea Change". *Journal of Educational Technology* Volume: 28 Issue: 2

Smith, J., 1999, "Psychological aspects of programming language choice: Why is the choice of programming language so emotionally charged?". MIT URL: http://www.media.mit.edu/~jsmith/sas/languages4.html

Stroustrup, B., 1994, *The Design and Evolution of C++*. Addison-Wesley

Swanke, J., 1999, "Coding Forever? Think twice about how you want to spend the rest of your life". URL: http://www.acm.org/pubs/articles/j…/drdobbs/1999/9913/9913e/9913e.htm