Considering the Impact of Programming Languages on Software Longevity

Bridget Lane

George Mason University

Author Note

This paper was prepared for English 302, Section N05, taught by Professor Taciuch.

Considering the Impact of Programming Languages on Software Longevity

Over the years, programming languages have grown rapidly. These rapidly changing circumstances and expanding opportunities for programmers have led to new and innovative software that evolves quickly. As research has shown, an environment of quickly changing programming languages has affected the longevity and overall flexibility of software, making picking a language and predicting software trends a now crucial part of software development. The ability to predict software changes and to be flexible with languages has a large impact on a company’s product. Alternatively, some believe that rather than focusing on software prediction, long-term academic commitment to the languages of modern technology is the solution to long-term maintenance and upkeep. Current research demonstrates that government-based companies, who frequently invest in their own programming languages or continue to use older languages such as COBOL, have a disadvantage over smaller, start-up companies who use the latest and academically-taught languages and technologies. In order to figure out how to reverse this trend and modernize these antiquated languages, one can consider reforming current languages to enhance adaptability, investing more in predictive software models, and higher-education curriculum changes that not only incorporate older languages, but also put a stronger emphasis on backwards-compatibility.

**Background**

Languages have developed and separated over syntax and logic throughout the history of programming, and this extremely fast language growth has impacted how software is developed. In some cases, language growth has progressed faster than hardware growth. In his article “Programming your quantum computer,” Brian Hayes, senior writer for the journal American Scientist, talks about how languages have already been developed for quantum computers. Although the hardware itself has yet to exist, programmers can already write code for these non-existent computers using languages such as QLC and Quipper, and languages continue to emerge at a fast pace. Hayes’ discussion demonstrates that throughout the existence of computer science, new languages will continue to be developed and increase in popularity and functionality. Yaofei Chen, Rose Dios, Ali Mili, Lan Wu, and Kefei Wang, authors of “An empirical study of programming languages,” talk about how programming languages directly correlate with software engineering trends. The authors demonstrate this by emphasizing programming languages’ deeply ingrained history in computer science and their wide diversity. By examining several different factors of programming languages and comparing these factors with historical data, they are able to apply this knowledge towards a predictive model of software trends. The fact that this model was developed from data almost entirely from programming languages demonstrates the strong impact that languages have on software development. Progression in languages and software trends are strongly linked.

At the point where it is understood that languages have a prominent effect on software, selecting a language becomes a crucial point in a company’s development. In “The semicolon wars,” Brian Hayes talks about how languages are different in part because they are created for separate audiences or applications. In “Programming your quantum computer,” Hayes presents a drastic example of this when talking about how quantum computers will change the programming and computing world as we know it. The influence of this technology has the potential to completely change programming languages as we know them. This kind of impact could cause an overhaul of industry on a large scale. Likewise, consistently patching vulnerable code will work for a time being, but eventually old and obsolete data structure and coding practices have to be redone. As languages progress and re-release, businesses that don’t adapt may fall behind. In “Estimating software readiness using predictive models,” Tong-Seng Quah, a researcher at a university in Singapore, focuses on quantifying software readiness. When attempting to assess the quality of software, one significant factor is reliability. Picking a language that will grow with future software engineering trends is an important part of business development, and will protect software from vulnerabilities and bugs.

A language, when used correctly, is much more of a powerful of a tool than a language used incorrectly. However, languages are not always selected purely for their power. In “An empirical study of programming languages,” discussed earlier, the authors compile data on the usage of several languages. Their data shows results of the respondent’s grassroot support for a language, primarily used language for school, and primarily used language for work. There is a clear correlation in the data between languages with strong grassroots support and languages more commonly used in business. Favorite languages go on to be used more often by companies in the workforce, while less popular languages are rarely used. Therefore popularity and personal choice seems to play a role in businesses selecting a language of primary use, in addition to functionality.

Looking at the history of languages, we can see that languages invested in by large-scale government corporations do not dominate compared to languages developed in obscurity by smaller researchers. At times, those languages developed by government organizations were highly specialized, advanced, and appropriate for use, yet still found little success. “A quantitative model for software engineering trends,” written by Latifa Rabai, Yan Bai, and Ali Mili, presents some startling historical facts about language development. Ada, a language with the support of the United States’ Department of Defense, saw little success or impact in the field of computer science. Another language, Prolog, was created with support from several governments around the world as a product of global research in an effort to advance computing, as saw little to no success. In comparison, Pascal, C, and Java were all developed by either one or two independent researchers, and saw huge success and had a vast impact. Overall, the languages that had a better flow to them or were overall liked more became much more popular, even if one could argue they were not as highly specialized or powerful in certain areas as the less impactful ones. Governmental agencies invest in higher-level languages in part to have some control over or understanding of the future of software. However, because a language has higher-level features does not always mean it will be popular or commonly used. In “The semicolon wars,” Hayes claims we have “yet to discover the perfect language,” although many computer scientists, he believes, would argue otherwise. He believes that many of them would argue strongly that their favorite languages are the perfect language. The result of this behavior is in part why companies are more inclined to work with newer and more popular languages overall. Companies with the ability to do this could stand to have an edge over companies who don’t.

**Discussion and Conclusion**

One strong case of the disadvantages faced by companies using antiquated languages is the continued use of COBOL in many branches of the Unites States government. Nicole Blake Johnson (2014), a technology journalist from Washington, DC, wrote about what she called “The Government’s COBOL Conundrum.” While many government agencies are looking to switch to a modern language such as Java, a switch like this is extremely costly. This is no surprise, considering that the Social Security Administration (SSA) has around sixty-million lines COBOL in use, in systems that have been around for decades and are functioning fine (U.S. GAO, 2014). The real difficulty in keeping these systems around, however, is finding someone who is capable of writing the code. In a poll of one-hundred and nineteen universities worldwide, Micro Focus found that while thirty-two percent of the universities introduced more than thirty Java developers to the job market, only five percent of universities introduced a comparative number, over thirty, of COBOL developers to the job market (Jackson, Rahman, & Stiff, 2013). Another statistic showed that in the United States, only twenty percent of universities require their computer science students to take a class that uses COBOL before graduation, and even more shockingly, only about twenty-five percent of universities are offering any COBOL courses at all (Konkel, 2013). As Johnson’s article noted, the United States’ Office of Personnel Management’s 2014 “Strategic information technology plan” reflected these concerns, stating that costs are anticipated to increase ten to fifteen percent each year “as personnel with the necessary coding expertise retire and cannot easily be replaced” (p. 47).

The COBOL language is still functioning well. In fact, Micro Focus’ Scot Nielsen says, “COBOL still remains the most readable programming language ever invented, and it maintains that characteristic that no other language has decided is important” (Konkel, 2013).

|  |  |
| --- | --- |
| ***COBOL:*** | ***Java:*** |
| IDENTIFICATION DIVISION.  PROGRAM-ID. HELLO-WORLD.  PROCEDURE DIVISION.  DISPLAY 'Hello world!'.  STOP RUN. | class HelloWorld {  public static void main(String[] args) {  System.out.println("Hello World!");  }  } |

*Figure 1*. Readability comparisonof business-centered COBOL and object-oriented Java using the common “Hello World!” program.

Regardless of this, the fact still stands that companies are gravitating towards modern languages. A company might use a cutting-edge language to attract employees, but what then happens when that language goes out of date? If these smaller businesses were to stick around for decades to come, they might find themselves in the same predicament government agencies using COBOL are in now.

One could argue that eventually, all languages will go out of date. For the future of big business, is large-scale language conversion an option? Currently, over seventy percent of business transactions require the use of COBOL (Konkel, 2013). Overall, the large scale and impact a language like this has had is immense and expensive to recreate. There are some examples of times when a conversion has gone correctly. With the help of Computech, the Customs and Border Protection Agency converted all of their COBOL code (about seven million lines) over to Java. But the cost was an exorbitant seven-hundred and fifty million dollars, and the changes had to be made incrementally. This is a lot of agony for a new language that, in a few decades, might also be out of date. The U.S. Postal Service considered doing the same thing, only to realize the job would be simply too large and complex. Instead, they’re focusing on methods to repurpose the COBOL code. Micro Focus, a company that wants to revitalize COBOL for the new generation, created a compiler that can repurpose COBOL code to run on Linux systems. The service is also looking into new ways to help the language have more of the features and amenities of modern-day programming languages, and to develop a COBOL training program for new programmers. This way, they can avoid the expense and risk of switching languages (Robinson, 2009). Jon Dittmer is the vice president and general manager of the defense sector at Array Information Technology, and former defense program manager, challenged with upgrading Defense Department’s COBOL system. While he recognizes the difficulty of conversion, he believes that “delaying the effort [of modernization] will only put more skeletons in the closet — and increase the size, scope and scariness of the modernization that’s required down the road” (Dittmer, 2013).

Looking at the research presented and Dittmer’s statement, it can be seen that computer science has an issue on its hands. As technology moves forward, should limited funding and research hours be applied towards backwards compatibility and training, or large-scale conversion and modernization? While an argument can be made for both sides, one could also present a third side. Time should be spent creating and reforming current languages so that they are adaptable, rather than expendable. When new features emerge in technology, new options for now-antiquated technology should be released alongside the newer features. By challenging computer scientists to think about not only new technology, but also old technology, backwards-compatibility will become a much more plausible and natural form of expansion.

While current research moves towards a predictive model of software trends, there is not yet one definitive or comprehensive model of software trends with regards to language. In order to stress the importance of backwards-compatibility, more research is needed that clearly demonstrates the natural consequences that may occur for companies that do not adapt or that continue to use antiquated languages without change. It is important to understand how accurate predictive models give advantages to companies that follow them, in order to understand the importance of languages and the ability to be flexible in development. Predictive models need to be narrowed in order to develop a solid predictive model for programming language trends. With the consideration of the heavy role that languages have to play in software maintainability, flexibility, and longevity, predictive models are able to play a larger role in corporations from their inception. Considering languages more heavily in future models of software prediction will expand predictive models and help isolate factors of progression in languages, which will allow businesses to assess their weaknesses immediately upon picking a language.

Also to be considered is the emerging consensus that education needs to change. However, if a larger emphasis was placed on backwards-compatibility and secure coding practices, perhaps the stigma that latest technology is also the most important technology would change. In addition to incorporating old but common languages such as COBOL in curriculum, the educational world needs to emphasize taking current trends and applying them to old language practices, in order to help modernize code and ultimately make our investments as programmers long-lasting. While predictive models are a solution that needs to be invested in, without proper reinforcements of older technology, the COBOL problem will perpetuate itself in multiple instances of today’s modern code in the future. With a smaller emphasis on conversion and a higher emphasis on adaption, predictive models, and education, government-based companies can begin to reverse the disadvantage they have over smaller and more modern start-up companies.

References

Arfa Rabai, L., Bai, Y., Mili, A. (2011). A quantitative model for software engineering trends. *Information Sciences, 181*(22)*,* 4993-5009.

Chen, Y., Dios, R., Mili, A., Wu, L., Wang, K. (2005). An empirical study of programming languages. *Software, IEEE, 22*(3)*,* 72-79. doi: 10.1109/MS.2005.55

Dittmer, J. (2013). The scary truth about modernization. FCW: The Business of Federal Technology. Retrieved from http://fcw.com/articles/2013/12/09/comment-legacy-it-skeletons.aspx

Hayes, Brian. (2006). The semicolon wars. *American Scientist, 94*(4)*,* 299. doi:10.1511/2006.60.3489

Hayes, Brian. (2014). Programming your quantum computer. *American Scientist, 102*(1)*,* 22. doi:10.1511/2014.106.22

Jackson, A., Rahman, N., & Stiff, L. (2013). Academia needs more support to tackle the IT skills gap. Micro Focus. Retrieved from http://www.microfocus.com/about/press/pressreleases/2013/pr070320131001.aspx

Johnson, N. B. (2014). The government’s COBOL conundrum. FedTech. Retrieved from http://www.fedtechmagazine.com/article/2014/06/governments-cobol-conundrum

Konkel, F. (2013). COBOL still integral to government systems. FCW: The Business of Federal Technology. Retrieved from http://fcw.com/articles/2013/12/12/cobol-legacy.aspx

Quah, Tong-Seng. (2009). Estimating software readiness using predictive models. *Information Sciences, 179*(4)*,* 430-445.

Robinson, B. (2009). COBOL remains old standby at agencies despite showing its age. FCW: The Business of Federal Technology. Retrieved from http://fcw.com/articles/2009/07/13/tech-cobol-turns-50.aspx

United States Government Accountability Office. (2014). SSA needs to address limitations in management controls and human capital planning to support modernization efforts. Retrieved from http://www.gao.gov/assets/670/663057.pdf

United States Office of Personnel Management. (2014). Strategic information technology plan. Retrieved from http://www.opm.gov/about-us/budget-performance/strategic-plans/strategic-it-plan.pdf