English and Programming: Factors Influencing the Widespread Use of English in the Software Development Field

A Research Paper submitted to the Department of Engineering and Society

Presented to the Faculty of the School of Engineering and Applied Science
University of Virginia • Charlottesville, Virginia

In Partial Fulfillment of the Requirements for the Degree
Bachelor of Science, School of Engineering

Christopher Hamilton
Spring 2022

Advisor
Kent Wayland, Assistant Professor, Department of Engineering and Society

Introduction

The rapid expansion of computer capabilities has resulted in an unprecedented lifestyle change for humanity. With technology's rapidly increasing role, computers have become embedded in modern life. Computer programming is becoming an essential skill for everyone to learn; however, the ability to learn coding is not accessible to everyone. With all of the most popular programming languages being implemented in English, the programming field is strongly rooted in the English language. The field is skewed toward those with mastery of English, and those who cannot speak English are put at a disadvantage. If the programming field continues with this English bias, diversity among developers will remain severely limited. Not only does this serve as an entrance barrier for many, but it limits the perspectives in software design. While the programming field certainly won't change overnight, the English bias in the field should be recognized by those who influence the development of the field. By doing so, they can make better decisions that influence the future of the programming field.

In this paper, I will first look through the history of computer programming to understand how the field of programming developed into an English-dominated field. Analyzing the history of the field will highlight the factors that contributed to English dominance seen today and will provide insight into the factors that keep the field from changing. I will then analyze how the heavy use of the English language affects the modern programming field. The combination of the historical analysis with the analysis of the modern field will provide a better understanding of the programming field as a sociotechnical system.

Historical Analysis of the Programming Field

To develop an understanding of the history of software, I investigated the work of Martin Campbell-Kelly, a professor who specializes in the history of computing. From his accounts, I created criteria to guide my research on programming history. I have decided not to look into historical accounts from the early stages of the computing industry because they are mostly technical and lack the hindsight that modern historical accounts offer (Campbell-Kelly, 2007). I have also chosen not to focus on primary accounts of the software field, such as industry reports. The field of software history has a considerable amount of breadth that I would not be able to address in its entirety within the scope of this paper. For this reason, I have decided to focus on programming history primarily based on the development of major programming languages. The study of programming languages is essential for understanding the impact of speaking languages and will offer a manageable lens through which to look at the history of the software field. With the above considerations in mind, I decided to use *1957–2007: 50 Years of Higher Order Programming Languages* as the main account for historical developments in the software industry. It both accounts for the history of the field over a long period and focuses on the programming languages. I have also selected a handful of other sources that meet the same criteria to supplement this account of history.

I will analyze the history of the software field through the framework of Technological Momentum. According to this theory, technology is free to be shaped by social factors in the earlier stages of its development. As a technology becomes widely accepted, it develops momentum, and it becomes more costly to make changes to the technology. Using this idea, I will observe how the use of the English language has followed the framework of technological momentum throughout its history. For the purposes of my analysis, I will consider technological momentum to be developments that have English aspects or that would make it particularly difficult for the field to transition away from English. Using this framework will offer insight into how the English-dominated programming field was developed, and it will provide a way to analyze the current programming field (Molella, 2005).

In the early days of computing, programming was done at the machine-level. The machine-level is considered very close to how a computer physically operates and is thus relatively far from how the actual human brain works. As such, machine-level computation was largely rooted in numbers and mathematical calculations. Since this form of computation was so limiting, computer functionality was restricted to the automation of calculations that would be regarded as simple by today's standards. Because of these limitations, a programmer's native language has comparatively little impact on the way that people interact with early computers. As the technology of computer programming has developed, programming languages have developed to a much higher level. High-level languages are further abstracted from the true workings of the computer and are much closer to what would be considered as understandable as humans. For this reason, I will start my account of programming history when the first major developments in high-level programming languages occurred (Wirth, 2008).

High-level computing began its rise to prominence in the mid twentieth century. At this time, computing was restricted to predominantly the academic and national defense sectors. Keeping in mind the historical context of the 1960s, the world was in the heart of the Cold War, where the superpowers of the U.S. and the Soviet Union were competing for technological and military dominance over each other. Computation was at the forefront of many notable government endeavors that attempted to assert superiority over the opposing side, including examples such as space travel and military technology development. The Soviet Union is generally considered to have lagged behind in the computation front and historical accounts of their computation are far more limited than that of the western world (Malinovsky, 2010). For this reason, I will focus on the programming developments in the western industrialized world.

Since the field of high-level computing was undeveloped, there was relatively little technological momentum in the programming field by the early 1950s. As such, the field was open to be shaped by the social factors that surrounded it. With this context in mind, we can look at what is the first widely used, high-level programming language. Fortran is a programming language developed at IBM (Wirth, 2008). This language was designed for use with the IBM 701 computer and was intended to make the programming of mathematical computations easier for the programmer. In the lower-level languages that were present before Fortran, there was very little abstraction from the actual workings of the computers, so programmers had to operate computers by controlling the actual logic gates of which the computer was composed (Lovrencic et al., 2009). To allow for greater programming capabilities meant to move the means of communication between programmer and computer closer to an understandable level for the human programmer. To effectively take advantage of the benefits of abstraction, abstractions must be consistent and complete models of the problems they represent (Wirth, 2008). As such, higher-levels of abstraction are more reliant on human language as a precise form of communication. Because Fortran was the first high-level language to see widespread use, it represents an early instance of English technological momentum in the programming field.

Fortran was a giant leap forward in the programming field, but the language itself had many limitations and was notably difficult to use (Lovrencic et al., 2009). As a result, several programming language projects sought to address the shortcomings of the Fortran language. While there was a handful of notable other follow-ups to Fortran, COBOL is regarded as the second widely accepted high-level programming language. In 1959, a committee of major U.S. computer manufacturers and branches of the U.S. defense sector created a new language. Because COBOL was created for use in administrative and finance systems, it was designed to have easy readability (Lovrencic et al., 2009). With COBOL programs designed to read like sentences, the programming language was strongly rooted in the English language. With the prominence of both Fortran and COBOL throughout the 1960s and 1970s, the programming field established a solid English foundation.

While the early languages above were experiencing widespread use, the C programming language was developed by Bell Labs and was released in 1972 (Lovrencic et al., 2009). It is notable that Bell Labs was a research branch of AT&T, who at the time was a massive government-regulated monopoly (Gertner, 2013). As such, the innovations that came out of Bell Labs were designed for English use. The C language, along with other influential languages of the 1980s like Pascal and Ada, allowed for a major evolution in the field of programming. During this period, the programming field started to develop capabilities for more abstract problem solving. A greater base of knowledge in the field and these more sophisticated languages allowed for greater applications of software (Wirth, 2008). As the field expanded in this way, human language became a much more of an essential component. Given that the field was still dominated by the United States at the time, this development served to further cement the base of the English language in programming.

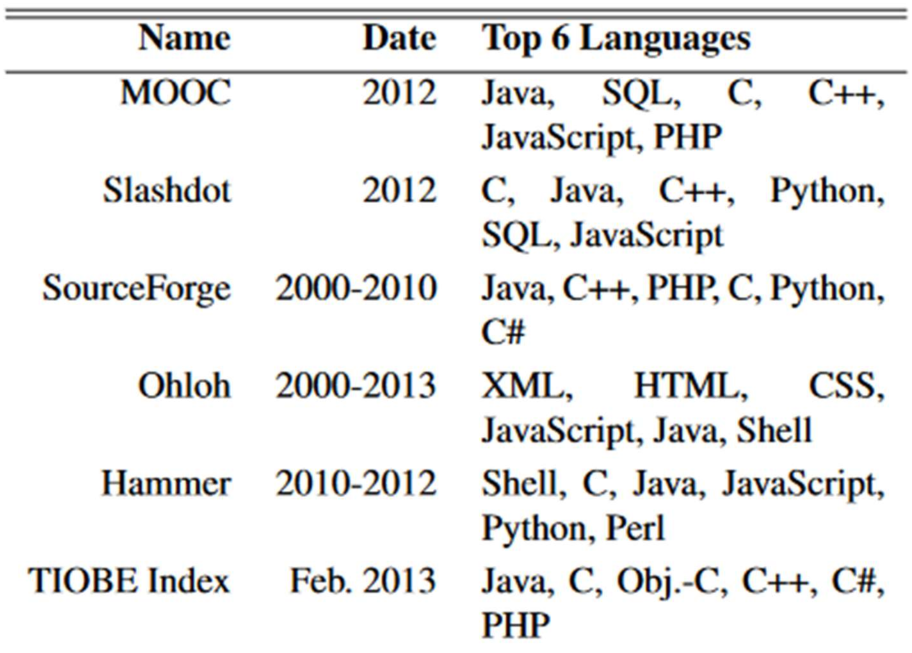
In the late 20th century, a large amount of technological momentum developed due to the rise of high-level programming languages. C++ was released in the 1980s and it was one of the first object-oriented languages to become widely used (Lovrencic et al., 2009). Along with other prominent object-oriented languages like Java, object-oriented programming came to dominate the software industry. The previous languages discussed until this point have been procedural programming languages. While procedural languages are very useful for simple computing tasks, they lack the capability to clearly represent the highly abstract and complex problems. Object-oriented programming languages are structured such that they can better represent many complex real-world problems. Along with the continual progression of hardware capabilities, the advent of object-oriented languages allowed for even higher levels of abstraction in programming (Lovrencic et al., 2009). These higher levels of abstraction forced developers to deal with greater levels of complexity and use detailed representations of problems (Bernholdt et al, 2004). The higher level of detail required for object-oriented designs resulted in programmers relying more on human communication to accurately express abstractions. As such, the advent of object-oriented languages served to further root the programming field in the already dominant English language.

At the turn of the century, the programming field was firmly solidified around the English language despite the internet allowing greater access to the computing world. Even programming languages developed in countries where English wasn't the native language were implemented in English. Python and Ruby are two of the most popular languages today, and they were developed in the Netherlands and Japan, respectively (McCulloch, n.d.). Despite this fact, both languages are fully implemented using the English language. If the developers of Python or Ruby used their native language, the programming languages would have little chance of succeeding in the English-dominated programming field. This serves as a prime example of the technological momentum keeping the field of programming centered around English.

The Modern Programming Field

To build on the analysis from the last section, I will look at how the large amount of technological momentum restricts changes in the field, as well as the impact of English dominance in the field of programming. Research in the area of language's impact on programming is limited, and many issues in the field have not been thoroughly investigated (Alaofi, 2020). Given that the learning of computer programming depends on people's prior experiences and abilities, it is difficult to isolate one's language mastery as the variable controlling their learning outcomes (Nnass, 2020). With this in mind, I have aggregated the findings of a handful of recent studies on the effect of English dominance in the programming field. I believe that this represents the best-available information at the time of writing this paper. For the purposes of this paper, I will consider the modern-day programming field to be the 21st century.

Looking at the modern field of programming, we can see the manifestation of the technological momentum that was highlighted in the previous section. English dominance of the field is particularly noticeable in the programming languages that are widely used today. Because a small number of languages account for the large majority of programming use, looking at the most popular languages gives a good representation of general language use (Meyerovich & Rabkin, 2013). Shown in figure 1 are the survey results of the most popular programming languages used in the software industry around 2010. There is a considerable amount of overlap between the most popular languages, and all of the listed languages are implemented in English (Meyerovich & Rabkin, 2013).



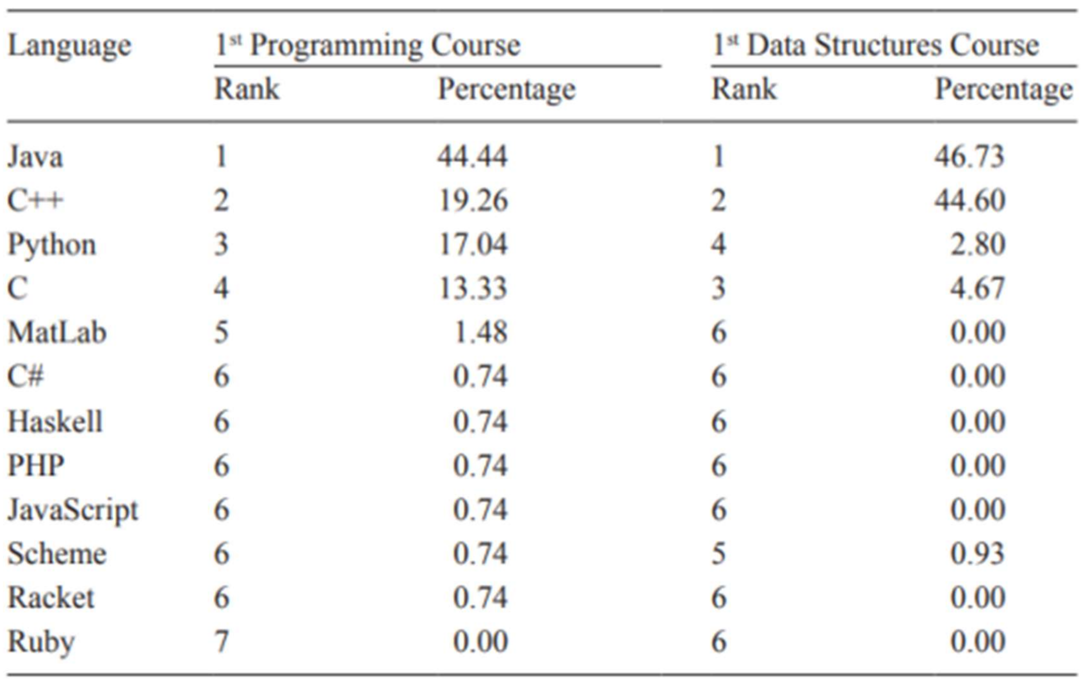
Name	Date	Top 6 Languages
MOOC	2012	Java, SQL, C, C++, JavaScript, PHP
Slashdot	2012	C, Java, C++, Python, SQL, JavaScript
SourceForge	2000-2010	Java, C++, PHP, C, Python, C#
Ohloh	2000-2013	XML, HTML, CSS, JavaScript, Java, Shell
Hammer	2010-2012	Shell, C, Java, JavaScript, Python, Perl
TIOBE Index	Feb. 2013	Java, C, Obj.-C, C++, C#, PHP

Figure 1. Most popular programming languages. This figure shows aggregated survey data for what are considered to be the most popular programming languages.

To better understand the impact of English prevalence in programming languages, it is important to look at how the English use in programming languages affects programmers. At their core, programming languages are made up of predefined keywords that a developer uses to write a program. The choice of these keywords is functionally arbitrary, but words such as “for” and “while" are commonly used because of their English meaning (Nnass, 2020). By assembling a language of meaningful keywords, programs written in that language are easier for developers to comprehend. Those who cannot easily interpret the meaning of these words will not be able to take advantage of the intrinsic readability of programs that the use of meaningful keywords offers. In addition to the keywords of a programming language, outputs of computer programs are commonly represented in English. Error messages are an example of information generated from the execution of a computer program, and they are responsible for communicating important information when verifying the functionality of code. In a study conducted at CQUniversity, many Libyan students indicated that they had difficulty interpreting error messages (Nnass, 2020). As the Libyan students did not speak English as their primary language, this would indicate that someone's English skills will affect their ability to use these messages effectively.

Another factor that skews a programming language's use toward a particular speaking language is the documentation that accompanies a language. Documentation serves as a guide to understanding how to use a particular programming language, and proper documentation is essential to program effectively in that language. Nearly all the most popular programming language support materials are written primarily in English (Guo, 2018). Trying to add support for other languages would add a high degree of complexity to programming language. Keeping software projects as simple as possible is essential because complexity greatly increases the chances for costly bugs (Wirth, 2008). As such, it makes the most sense to keep programming languages in familiar territory and refrain from transformative changes. With this in mind, it seems an impossible task for developers to add support for languages other than English. This serves as yet another factor preventing the field of programming from transforming to be more inclusive of other speaking languages.

Beyond the languages and support materials being written in English, the fact that most programming classes are taught in English further roots the programming field in use of the English language (Veerasamy, 2014). An example of this is the Computer Science curriculum in India being predominantly taught in English. This can also be seen in the commonalities in the programming languages used in university-level classes. Looking at programming language usage from college classes in 2013, we can see that the curriculum is largely dominated by four popular languages (Ben Arfa Rabai et al., 2015). For the University, it makes sense to focus the curriculum on prominent programming languages because it allows for a stable curriculum and better prepares the students for the real-world programming field.



Language	1st Programming Course		1st Data Structures Course	
	Rank	Percentage	Rank	Percentage
Java	1	44.44	1	46.73
C++	2	19.26	2	44.60
Python	3	17.04	4	2.80
C	4	13.33	3	4.67
MatLab	5	1.48	6	0.00
C#	6	0.74	6	0.00
Haskell	6	0.74	6	0.00
PHP	6	0.74	6	0.00
JavaScript	6	0.74	6	0.00
Scheme	6	0.74	5	0.93
Racket	6	0.74	6	0.00
Ruby	7	0.00	6	0.00

Figure 2. Programming language use in universities. This figure shows survey data of programming language use in college-level programming classes from 2013.

Teaching programming in English places an increased burden on students lacking strong English skills (Soosai Raj et al., 2018). Non-native English speakers face the challenge of mentally translating content to their native language, increasing their cognitive load and decreasing their comprehension of the material (Guo, 2018). To substantiate the claim that there are significant difficulties pursuing education outside one's native language, data mining has shown that non-native English speakers have high drop-out rates in English undergraduate education (Kumar & Pal, 2012). English dominance in the classroom is yet another example of the immense technological momentum in the field of programming.

Conclusion

This paper has looked at the history of the programming field regarding the heavy use of the English language. As a result of many historical factors, the programming field began as a field rooted in English and has remained as such for the entirety of its history. As the field has developed, so too has its use of the English language, resulting in a programming field that is heavily reliant on English. The modern field of programming is dominated by English-based programming languages, and support and instructional materials are also predominantly implemented in English. Coupled with the historical English dominance in the field, this shows the large amount of technological momentum that the English language has in the field of programming.

These findings bring forth major questions for the programming field moving forward. Firstly, what should be done about this English dominance in the programming field? Viewing this from the perspective of distributive justice, it would be best for the career field to be as inclusive of different backgrounds as possible. This perspective would dictate that it would be ethical to make sweeping changes to the field of programming as it stands today, but this suggests a second and perhaps more important question. What can feasibly be done to change the English dominance in the field? Despite the good reasons for trying to change the programming field, it is highly questionable whether large-scale changes are even realistic. Sweeping changes would be highly costly, both from a monetary standpoint and the overall quality of software projects. Such implications could harm our modern society that is so reliant on computer technology, especially in safety-critical applications of software. Therefore, it is certainly not cut and dry as to whether attempting to change the field of software development in this way would have a net-positive impact. Considerations such as these should be acknowledged by those who influence the continuing development of the programming field, such as programming languages developers, educational institutions, and programming researchers. Doing so may help guide them toward making the best possible programming field moving forward.

References

Alaofi, S. (2020). The Impact of English Language on Non-Native English Speaking Students' Performance in Programming Class. *Proceedings of the 2020 ACM Conference on Innovation and Technology in Computer Science Education*, 585–586. https://doi.org/10.1145/3341525.3394008

Ben Arfa Rabai, L., Cohen, B., & Mili, A. (2015). Programming Language Use in US Academia and Industry. *Informatics in Education*, *14*, 143–160. https://doi.org/10.15388/infedu.2015.09

Bernholdt, D. E., Nieplocha, J., & Sadayappan, P. (2004, February). Raising level of programming abstraction in scalable programming models. In IEEE International Conference on High Performance Computer Architecture (HPCA), Workshop on Productivity and Performance in High-End Computing (P-PHEC) (pp. 76-84). Los Alamitos: IEEE Computer Society.

Campbell-Kelly, M. (2007). The History of the History of Software. *IEEE Annals of the History of Computing*, *29*(4), 40–51. https://doi.org/10.1109/MAHC.2007.4407444

Gertner, J. (2013). *The Idea Factory: Bell Labs and the Great Age of American Innovation*. Penguin.

Guo, P. J. (2018). Non-Native English Speakers Learning Computer Programming: Barriers, Desires, and Design Opportunities. *Proceedings of the 2018 CHI Conference on Human*

Factors in Computing Systems, 1–14. https://doi.org/10.1145/3173574.3173970

Kumar, S., & Pal, S. (2012). Data Mining: A Prediction for Performance Improvement of EngineeringStudents using Classification. *World of Computer Science and Information Technology Journal*, *2*, 51–56.

Lovrencic, A., Konecki, M., & Orehovački, T. (2009). 1957-2007: 50 Years of Higher Order Programming Languages. *Journal of Information and Organizational Sciences*, *33*, 79–150.

Malinovsky, B. N. (2010). *Pioneers of Soviet Computing* (2nd ed.). https://www.sigcis.org/malinovsky_pioneers

McCulloch, G. (n.d.). Coding Is for Everyone—As Long as You Speak English. *Wired*. Retrieved September 27, 2021, from https://www.wired.com/story/coding-is-for-everyoneas-long-as-you-speak-english/

Meyerovich, L. A., & Rabkin, A. S. (2013). Empirical analysis of programming language adoption. *Proceedings of the 2013 ACM SIGPLAN International Conference on Object Oriented Programming Systems Languages & Applications*, 1–18. https://doi.org/10.1145/2509136.2509515

Molella, A. (2005). HUGHES ON TECHNOLOGY. *Minerva*, *43*(1), 113–117.

Nnass, I. (2020). *Identifying and solving issues with acquiring skills in computer programming for non-English speakers* [Thesis, CQUniversity]. https://figshare.com/articles/thesis/Identifying_and_solving_issues_with_acquiring_skills_in_computer_programming_for_non-English_speakers/13406825/1

Soosai Raj, A. G., Ketsuriyonk, K., Patel, J., & Halverson, R. (2018). *Does Native Language Play a Role in Learning a Programming Language?* 417–422.

https://doi.org/10.1145/3159450.3159531

Veerasamy, A. K. (2014, February 14). *Teaching English based programming courses to English learners/non-native speakers of English*.

Wirth, N. (2008). A Brief History of Software Engineering. *IEEE Annals of the History of Computing*, *30*(3), 32–39. https://doi.org/10.1109/MAHC.2008.33