Approaching polyglot programming: what can we learn from bilingualism studies?

Rebecca L. Hao
Department of Computer Science and Department of Linguistics, Harvard University, USA
rhao@college.harvard.edu

Elena L. Glassman
Department of Computer Science, Harvard University, USA
glassman@seas.harvard.edu

Abstract

Today's programmers often need to use multiple programming languages together, enough that this practice has been given the name "polyglot programming." However, not much is known about how using multiple programming languages affects programmers, despite its increasing ubiquity. If we want to better design programming languages and improve the productivity of programmers who program in multiple programming languages, we should seek to understand the user in this context: we need to better understand the impact that polyglot programming has on programmers. In this paper, we pose several open research questions to begin to approach this question, drawing inspiration from psycholinguistic studies of bilingualism, because despite the differences between natural languages and programming languages, the questions considered in natural language bilingualism studies are relevant to programming languages, and the existing findings may prove useful in guiding our intuitions, methods, and priorities as we begin to explore this topic. In particular, we pay close attention to the implications that code switching (switching between languages within a conversation) and interferences (ways an unintended language may influence one's use of an intended language) may have on our understanding of how using programming languages may impact a programmer.

2012 ACM Subject Classification Human-centered computing → Human computer interaction (HCI); Software and its engineering → General programming languages

Keywords and phrases Programming languages, polyglot programming, bilingualism

Digital Object Identifier 10.4230/OASIcs.PLATEAU.2019.

1 Introduction

Programmers today are often expected to use multiple programming languages at once, engaging in *polyglot programming*. Perhaps they are developing for the web, and using some mixture of HTML, CSS, Javascript, SQL, and others. They could be using a virtual platform like JVM or .NET, using a number of programming languages and libraries together. Maybe they require a more expressive, higher-level scripting language or domain specific language, embedding it into a lower-level one with better run-time performance like C or C++. On a project level, they could build separate programs and let them interact and send data to each other using pipes, cross-compile one language into another, or use a language's ability to interface with other languages. Even within a single file, they may need to mix programming languages [1].

The advantages of polyglot programming include the cost-effective ability to reuse code already written in other languages, and using programming languages that are better suited to one's goals. After all, programming languages are not created equally – each has its characteristics and strengths, and is more useful in certain domains and tasks than others.

Given the commonality of programmers writing code in multiple programming languages, we think that it is important to know more about the impact that using multiple programming languages has on the programmer, and how the knowledge and use of multiple programming

languages affects the code the programmer writes. Intuitively, we may think that knowing multiple programming languages strengthens one's understanding of programming concepts, or that switching between languages is difficult and may incur some cognitive cost, but how well do these intuitions hold and how can we go about studying them?

Though programming languages and natural languages have their differences, there are still a number of compelling parallels. One such parallel that is relevant to polyglot programming is that there are many polyglots of natural language, who use different languages in different domains. Similarly to programming languages, natural language speakers may switch languages at multiple scales, a practice known as code-switching, substituting words and phrases in one base language with those of another, or switching languages altogether depending on who they are speaking to and in what context.

There have been a number of interesting findings in psycholinguistics, the study of the relationship between linguistic behavior and psychological processes, that deal with the learning, use, and switching of multiple languages [7, 8, 9]. In this paper, we examine how psycholinguistic studies of natural language bilingualism may relate to that of programming languages, and argue that a psycholinguistic lens of polyglot programming may be useful towards guiding the questions that we ask, the priorities that we set, and the methodologies that we use. We choose to focus specifically on the case of the *use* of multiple programming languages, which involves code switching (switching between two language systems), and the direct impact that knowledge of multiple programming languages has on programming. However, this psycholinguistic lens may also prove to be useful in other polyglot programming topics, including programming language learning and the effects that the knowledge of multiple programming languages has on programming concept understanding.

In this way, by learning more about how programmers use and are affected by their use of multiple programming languages, we can inform programming language design in the context of its use with other languages, and also inform the practice of using multiple programming languages, working towards increasing programmer productivity and efficiency.

2 Related work

2.1 HCI for programmers

HCI methods have already been used to examine programming languages and environments. This work seeks to better use the knowledge, principles, and methods of human computer interaction (HCI) in programming language design, showing that gathering data on and considering programming language usability leads to better design [15, 17]. Similarly, we seek to understand the impact of knowing and using multiple programming languages on users: the programmers.

2.2 Related work in programming languages

Work in the psychology of programming and the HCI-based approach toward programming language design has made major strides. For example, there are investigations of how a programmer's experience level impacts how they reason about, read, and/or debug programs [6, 13, 21]. However, the question of polyglot programming is less thoroughly explored.

Most of the work that addresses multiple languages focuses on the learning and teaching of programming languages. This work includes justifying and exploring the impact of using specific languages (like Python or Java) in introductory computer science courses [10, 11], and effective methodologies for learning subsequent languages [16, 19].

There is some literature surrounding the design of multi-language systems like .NET that discuss ways to support polyglot programming and its design [14], but it does not focus on the effects that these multi-language systems have on programmers and their productivity.

However, there is evidence that suggests that using multiple programming languages within a project may result in decreases in code quality. Kochhar et al [12] investigated the impact of multiple programming languages on software quality, applying a method of looking into code quality across Github projects proposed by Ray et. al [18]. In this study, Kochhar et al. looked for the effects of using different programming languages on the number of bug fix commits in a dataset of popular projects from Github. They found that in general, projects that used more programming languages were more bug-prone, especially for memory, concurrency, and algorithmic bugs, and that specific languages like C++, Objective-C, and Java were more bug-prone [12].

2.3 Related work in natural languages

Studies regarding natural language polyglots are more numerous. A central observation is that bilinguals operate using different language modes: the monolingual speech mode and the bilingual speech mode [8]. In the monolingual speech mode, the bilingual almost completely deactivates one language, while in the bilingual speech mode, they choose a base language but activate another language occasionally in the form of code-switching and borrowing [8]. This deactivation when a language is not in use involves prefrontal brain activity associated with cognitive control [2]. This cognitive cost may be due to task switching, as it has been shown that in word-reading and picture-naming, a greater time delay occurs after a task switch than a task repetition [20].

However, there is a nuance to *when* this cognitive cost occurs that may be relevant: a recent study suggests that *deactivating* a language requires cognitive control, while *activating* a new language may not [4]. This study used magnetoencephalography (MEG), and found that American Sign Language (ASL)-English bilinguals had increased brain activity in brain areas involved in cognitive control (the anterior cingulate cortex and dorsolateral prefrontal cortex) when deactivating a language, but not when activating one [4]. Additionally, because certain ASL-English bilinguals can sign and speak simultaneously, they also found that using both languages simultaneously did not necessarily incur a greater cognitive cost than producing one language on its own [4].

Psycholinguistic studies have shown that even though it is not conscious or intentional, code switching behavior is often rule-based: there are systematic patterns to how and when speakers code switch. Speakers may code switch by substituting single words from one language into a sentence with the grammar of the base language, or alternate between languages on sentence boundaries [7].

It has also been found that infants demonstrate cognitive load through involuntary pupil dilation and eye-tracking fixations when they listen to language switches. These effects, however, were reduced when going in a non-dominant to dominant language direction, and when switching languages when crossing sentence boundaries [5].

Additionally, even when in a monolingual speech mode and trying to speak one language (L_a), sometimes other languages that the bilingual knows (e.g. L_b) influence their speech in the form of interferences. This is known as cross-linguistic influence, and comes in the form of static interferences or transfers, which are more permanent traces of L_b in L_a (like with accents), and dynamic interferences, which are brief intrusions of L_b in L_a (like accidentally stressing the wrong syllable or momentarily using a word or grammatical structure in the other language). Static interferences are linked to a person's competence in L_b, while dynamic

interferences are more likely to occur when one is stressed, tired, or emotional. A language that is more closely related to the intended language has been found to have more of an influence on the intended language than one that is are more distant [9].

3 Open questions and potential approaches

We pose four guiding questions regarding the use of multiple programming languages that are inspired by the hypotheses, methods, and work within the psycholinguistic study of bilingualism.

1. How does knowledge of certain programming languages influence programming in another language?

Is there cross-linguistic influence [9] between programming languages? Do programmers ever accidentally use an unintended programming language syntax or concept while trying to program in a specific programming language? This could occur on a number of levels, taking on forms like: using a function word or symbol incorrectly in the desired language because of that symbol's use in another language, using an incorrect construct (e.g. for loop constructs look and act differently between C and Python), and approaching a function or program using logic that is clearly inspired by another language (e.g. trying a "Pythonic" way in a language where it is less conventional, or not even possible).

This may be challenging to isolate and quantify especially in existing projects and code, so studying programmers while they are programming may be of interest. We can observe and look out for errors that programmers run into and corrections that they make as they program, and their approaches to certain programming problems, paying particular attention to what programming language, if any, these errors come from and approaches are inspired by. This resembles the collection of linguistic data, which often involves utterances from native speakers, which are then analyzed by linguists for patterns and structures. To better understand the influences programming languages may have on each other, it may also be useful to investigate how much programmers seek equivalences between languages, for example, how often questions are asked about equivalent keywords or constructs from one language to another on forums like Stack Overflow.

2. Are certain kinds of languages more prone to influencing other kinds?

From there, it also seems important to determine which languages influence and are influenced by others, and if there are patterns in terms of the characteristics of the languages involved, or in terms of the relationships between the languages. For example, do similar languages influence each other more, as we find with natural language? Does knowledge of languages that use certain paradigms (e.g. static versus dynamic, functional versus object-oriented) have greater influence over the use of other programming languages?

In order to approach this question, we may need a more concrete way of describing a programming language polyglot's knowledge of their programming languages, since proficiency is likely closely related to the ability for a certain language to influence another. Natural language bilingualism studies have generally used two main factors to characterize bilinguals – language proficiency and language use – and have used these two variables as axes in a grid approach to map the language history of a polyglot [9]. In this approach, a language is represented on this grid as a datapoint of proficiency and use, and multiple grids may be used to show proficiency and use in specific domains, or across time. A similar approach

may prove useful for programming languages as well, because in this way we can investigate the time course of programming language learning, and have an improved ability to study how code quality and performance change with time, without requiring a study to be set up around a specific computer science course and structured learning. However, this may require that we determine what variables are relevant in programming language knowledge, and what data we would like to and can collect from our participants.

3. How and when do programmers code switch?

As programmer "code switching" seems to occur at several distinct levels of granularity, it may be useful to look into the occurrences of "code switching" at these different levels. An example delineation of these levels could be: switching within a line, switching for each line, switching at "logical blocks" of the code (e.g. where the author decided to stylistically include a new line for readability), and switching between files. We could learn more about the contexts in which code switching occurs in practice, to better understand the motivations of why programmers code switch in the first place.

From there, we can then look at the impacts of code switching on programmers. It may be useful to try out switching tasks inspired by psycholinguistic studies that, require speakers to name images in different languages in a controlled manner, measuring reaction time differences or neuroimaging studies [2]. The equivalent of having speakers name images could be having programmers name constructs for short code snippets. Alternatively, to study whether reading and understanding multilingual code incurs a cognitive cost, we could present programs in different languages and observe eye tracking or pupil dilation to indicate the presence of cognitive control.

4. What is the degree of cognitive control being exerted when switching between programming languages? Are there cases where we can reduce the cognitive cost of switching and thereby increase productivity?

Additionally, investigating programming in different languages using MEG may provide information of whether switching programming languages has a similar effect on prefrontal areas of cognitive control. This can be investigated when programming in a language-agnostic way (e.g. writing pseudocode), or when activating and deactivating languages (e.g. writing a program or code for a project that requires switching between programming languages).

Does the degree of cognitive control depend on the programmer's proficiency in a language? Is the cognitive cost higher in novices and lower in experts?

From there, inspired by natural language, we can investigate whether there are cases where the effects of code switching are reduced. Like how the effect is reduced in natural language when going from nondominant to dominant language and on sentence boundaries [5], for programming languages, are effects reduced in certain directions, like going from programming languages one is less familiar with to those one is more familiar with? Are effects reduced more on boundaries of a certain granularity, like between "sentences"? Like how cognitive cost is reduced in ASL-English bilinguals when using both languages simultaneously [4], is there a difference between when trying to program exclusively in one language or when actively harnessing knowledge from multiple languages simultaneously (e.g. pseudocode)?

4 Conclusion

We have proposed several open questions and next steps for the study of polyglot programming, inspired by findings and methodologies in natural language bilingualism. Even though programming languages likely do not rely on the same language faculty attributed to natural language and instead on different programming concepts, natural language bilingualism still serves as a useful model to guide foundational decisions to more concretely approach polyglot programming questions, and provide an intuition for possible phenomena related to the use of multiple programming languages.

This approach may also be extended beyond the *use* of programming languages, to areas like programming language learning and whether there are benefits of knowledge and use of multiple programming languages. For example, does knowing more languages improve your understanding of programming concepts? Bilingualism studies demonstrate a bilingual advantage in non-linguistic tasks that require cognitive control [3] – is polyglot programming beneficial to a programmer's general programming skills?

With an improved understanding of the use of multiple programming languages, we may be better equipped to design programming languages and tools given this growing polyglot programming landscape and improve programmer efficiency and polyglot programming workflows.

References

1 Why are multiple programming languages used in the development of one product or piece of software?, Apr 2018. URL: https://softwareengineering.stackexchange.com/questions/370135/why-are-multiple-programming-languages-used-in-the-development-of-one-product-or.

2 Jubin Abutalebi and David W. Green. Control mechanisms in bilingual language production: Neural evidence from language switching studies. *Language and Cognitive Processes*, 23(4):557–582, 2008. doi:10.1080/01690960801920602.

3 Ellen Bialystok. Cognitive complexity and attentional control in the bilingual mind. *Child Development*, 70(3):636–644, 1999. URL: https://srcd.onlinelibrary.wiley.com/doi/abs/10.1111/1467-8624.00046, arXiv:https://srcd.onlinelibrary.wiley.com/doi/pdf/10.1111/1467-8624.00046, doi:10.1111/1467-8624.00046.

4 Esti Blanco-Elorrieta, Karen Emmorey, and Liina Pylkkänen. Language switching decomposed through meg and evidence from bimodal bilinguals. *Proceedings of the National Academy of Sciences*, 115:201809779, 09 2018. doi:10.1073/pnas.1809779115.

5 Krista Byers-Heinlein, Elizabeth Morin-Lessard, and Casey Lew-Williams. Bilingual infants control their languages as they listen. *Proceedings of the National Academy of Sciences*, 114:201703220, 08 2017. doi:10.1073/pnas.1703220114.

6 Martha E. Crosby, Jean Scholtz, and Susan Wiedenbeck. The roles beacons play in comprehension for novice and expert programmers. In *PPIG*, 2002.

7 David W. Green and Li Wei. Code-switching and language control. *Bilingualism: Language and Cognition*, -1:1–2, 02 2016. doi:10.1017/S1366728916000018.

8 François Grosjean. The bilingual's language modes. *One mind, two languages: Bilingual language processing*, pages 1–22, 2001.

9 François Grosjean and Ping Li. *The Psycholinguistics of Bilingualism*. Wiley-Blackwell, Chichester, 2013.

10 Said Hadjerrouit. Java as first programming language: a critical evaluation. *SIGCSE Bulletin*, 30:43–47, 1997.

11 Tony Jenkins. The first language - a case for python? *Innovation in Teaching and Learning in Information and Computer Sciences*, 3(2):1–9, 2004. URL: https://doi.org/10.

11120/ital.2004.03020004, arXiv:https://doi.org/10.11120/ital.2004.03020004, doi:10.11120/ital.2004.03020004.

12 Pavneet S. Kochhar, Dinusha Wijedasa, and David Lo. A large scale study of multiple programming languages and code quality. In *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, volume 1, pages 563–573, March 2016. doi:10.1109/SANER.2016.112.

13 Yu-Tzu Lin, Cheng-Chih Wu, Ting-Yun Hou, Yu-Chih Lin, Fang-Ying Yang, and Chia-Hu Chang. Tracking students' cognitive processes during program debugging—an eye-movement approach. *IEEE Transactions on Education*, 59(3):175–186, Aug 2016. doi:10.1109/TE.2015.2487341.

14 Bertrand Meyer. Multi-language programming: how .NET does it. *Software Development*, 2002.

15 Jonathan Aldrich Michael Coblenz, Brad A. Myers, and Joshua Sunshine. Interdisciplinary programming language design. In *Proceedings of the 2018 ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software*, Onward! 2018, pages 133–146, New York, NY, USA, 2018. ACM. URL: http://doi.acm.org/10.1145/3276954.3276965, doi:10.1145/3276954.3276965.

16 Casey O'Brien, Max Goldman, and Robert C. Miller. Java tutor: Bootstrapping with python to learn java. In *Proceedings of the First ACM Conference on Learning @ Scale Conference*, L@S '14, pages 185–186, New York, NY, USA, 2014. ACM. URL: http://doi.acm.org/10.1145/2556325.2567873, doi:10.1145/2556325.2567873.

17 John F. Pane, Brad A. Myers, and Leah B. Miller. Using hci techniques to design a more usable programming system. In *Proceedings IEEE 2002 Symposia on Human Centric Computing Languages and Environments*, pages 198–206, Sep. 2002. doi:10.1109/HCC.2002.1046372.

18 Baishakhi Ray, Daryl Posnett, Vladimir Filkov, and Premkumar Devanbu. A large scale study of programming languages and code quality in github. *Proc. FSE 2014*, pages 155–165, 11 2014. doi:10.1145/2635868.2635922.

19 Karen P. Walker and Stephen R. Schach. Obstacles to learning a second programming language: An empirical study. *Computer Science Education*, 7(1):1–20, 1996. doi:10.1080/0899340960070101.

20 Florian Waszak, Bernhard Hommel, and Alan Allport. Task-switching and long-term priming: Role of episodic stimulus–task bindings in task-shift costs. *Cognitive Psychology*, 46:361–413, 2003.

21 Susan Wiedenbeck. Beacons in computer program comprehension. *International Journal of Man-Machine Studies*, 25(6):697 – 709, 1986. doi:https://doi.org/10.1016/S0020-7373(86)80083-9.