Becker, B. A. (2019). Parlez-vous java? bonjour la monde != hello world: Barriers to programming language acquisition for non-native english speakers. In 30th workshop of the psychology of program-ming interest group - PPIG '19 . Retrieved from: www.brettbecker.com/publications

Parlez-vous Java? Bonjour La Monde != Hello World: Barriers to Programming Language Acquisition for Non-Native English Speakers

Brett A. Becker
School of Computer Science
University College Dublin
brett.becker@ucd.ie

Abstract

Learning computer programming could and should be made easier. It is widely accepted that learning to program is fraught with challenges and the literature is not short of work that supports this view. There are many studies related to programming difficulties, barriers, and misconceptions as well as topics such as what language is best for learning and what techniques for teaching programming are most effective. It is often overlooked that globally, the majority of programming students are non-native English speakers. In addition to the barriers faced by all programming students, these non-native English speakers face a substantial class of additional barriers. This is because English is often the language upon which programming languages and their documentation are based, as well as the language of instruction and other environmental conditions.

There have been relatively few studies on the impact of human language on learning programming and the potential barriers this may cause. These barriers also span a wider range than may be obvious upon initial inspection. To complicate matters, natural language issues can add an additional layer of complexity to more universal barriers to learning. For instance it is well known that programming error messages present most novice programmers with difficulty. When these messages are in English as they most often are, any difficulties interpreting them and using them to produce error-free code are most likely compounded for non-native English speakers.

Particularly in a time when broadening participation in computing is a primary objective, the community can no longer afford to overlook the unique barriers faced by non-native English speakers who want to learn to program. This paper discusses these barriers, presents some questions to guide future research, and outlines the author's work-in-progress in the area.

1. Introduction

There have been few studies on the impact of human language on learning programming (Guo, 2018) and the challenges faced by non-native English speakers when learning how to program are poorly represented in the literature (Becker, 2015). This does not mean that it is not a very important area for research. It is likely that there are far fewer programmers whose native language is English than those who are non-native English speakers based on the fact that 95% of the world's population does not have English as their first language (Guo, 2018). However almost all programming languages are designed using English (Veerasamy & Shillabeer, 2014) and most likely the majority of resources and documentation are as well (Guo, 2018; Li & Prasad, 2005). These points alone provide significant justification to studying the differences between how native and non-native English speakers program, and learn to program.

It is also important to note that the treatment of programming languages as similar to natural languages is being discussed and acted upon by many, possibly more outside academic and educational communities than within. It seems that many believe that learning programming is more important than learning foreign natural languages and importantly there is also a reported difference in the degree to which people of different genders believe this.[1] There are also several political movements underway in the

[1] https://www.teachingpersonnel.com/news/people-would-rather-learn-coding-than-a-foreign-language--62462135356

United States where programming languages may be categorised as a foreign language in curricula[2] and counted as foreign languages for college entrance requirements[3]. Less than two years ago Tim Cook remarked "If I were a French student and I were 10 years old, I think it would be more important to learn coding than English. I'm not telling people not to learn English – but this is a language that you can [use to] express yourself to 7 billion people in the world."[4] When global technology leaders talk people listen, and there is a serious issue with this message – it implies that English language ability and learning programming are not intricately related. This view neglects to address the fact that there is substantial evidence – some of it bordering on common sense – that those who don't speak English can be at a real disadvantage when it comes to learning programming compared to native English speakers.

It is not a goal of this paper to provide a comprehensive view of the work on how non-native English speakers learn to program. It is a goal of this paper to set out a discussion on some of the barriers that these students face to inform future work on overcoming these barriers. We also pose some questions that may guide future research on the relationship between programming and natural languages and on the barriers that non-native English speakers may face when learning to program. Finally we present some early work-in-progress in the area.

2. Natural Languages and Programming Languages

Larry Wall, the developer of Perl, whose has a background in linguistics stated that "there is a scale of how much a computer language resembles human language primarily based on how much context is involved".[5] Programming languages are not natural languages, however they are languages (albeit artificial, and most commonly written only) that are designed to convey instructions to a computer and therefore have a restricted vocabulary and tightly-defined specifications (Eastman, 1982). However a computer program can (and arguably should) also convey meaning to other humans (Tenenberg & Kolikant, 2014). Further, it would be surprising if programming languages designed by humans did not share characteristics of the natural languages used by the language designers (Naur, 1975).

The relationship between programming and natural languages is complex. It is accepted that parsing natural language by computational means is more difficult than parsing programming languages by the same means. This may indicate that the human parsing mechanism works by other means making it less suitable for parsing programming languages. This would not be surprising, as programming languages were designed to be easily parsed by computational means and natural language evolved along with human brains for millennia. Nonetheless the relationship between natural and artificial languages (and specifically programming languages) is not frequently studied but some work has been done. For instance, Tenenberg and Kolikant (2014) presented several views that relied on multiple established theoretical perspectives on social cognition and human communication, speculating that these may be crucial to understanding how people learn to program computers. Specifically, by casting computer programs as speech acts, they considered that novices learning to program might, can, and sometimes do rely upon their prior and often extensive experience as skilled natural language users. Along similar lines Eastman (1982) demonstrated that programming keywords can be formed using mechanisms analogous to those observed in English such as neologism formation. Miller and Settle (2019) also presented a relationship between natural language and programming in metonymy. We discuss these findings further in Section 4.1.

It is beyond doubt that programming languages and natural languages are related. The extent to which this is true is beyond the scope of this paper. However, even if weak, if this relationship exists to any extent, it is likely that one's native language affects how a programming language is learned. Regardless

[2] https://www.usnews.com/news/stem-solutions/articles/2016-10-13/spanish-french-python-some-say-computer-coding-is-a-foreign-language

[3] https://www.fastcompany.com/3042122/washington-bill-would-count-programming-as-a-foreign-language-on-college-apps

[4] https://qz.com/1099791/apples-tim-cook-says-coding-is-better-than-learning-english-as-a-second-language/

[5] https://bigthink.com/videos/why-perl-is-like-a-human-language

of the parallels one draws between programming and natural languages, it is accepted that programmers have to speak 'computerish' – we are able to 'speak' C, Pascal, SQL or even machine code – and it has been stated that humans learn a computer language using the same faculties as learning natural languages, in an intuitive manner, yet without a profound understanding of what is going on in our brains during this process.[6] There is also some fresh empirical evidence in this department when it comes to programming languages that supports this hypothesis using human brain studies. In Section 3 we discuss an fMRI study that has provided evidence that code comprehension stimulates the same areas of the brain that natural languages do.

Unlike natural languages which can be quite forgiving due to their ambiguity (and the human ability to interpret that), modern high-level programming languages have a well-defined structure and syntax. Deviating from these specifications renders a program of little use. Therefore it is reasonable to hypothesise that non-native English speakers may be at more of a disadvantage compared to those fluent in English when it comes to code construction, code reading, and debugging. It is also possible that some of these means of interacting with programs may be more severely hampered than others, which requires that these factor be studied on a case-by-case basis. What is clear is that we are operating with high-level languages that are (hopefully) natural-language-like enough for people to use them freely without the need to spend large amounts of time just to figure out what the code should look like and at the same time exact enough for computers to parse it unambiguously.[6]

Another debate that we will not explore here but should be pointed out is 'teaching' or 'pedagogical' languages (Crestani & Sperber, 2010) vs. 'real' languages, which normally refers to languages that are used in industry. Interestingly, teaching languages may have parallels in natural languages when one considers Esperanto. Esperanto is an artificial natural language which has some features of a natural language – just as pedagogical programming languages have some features of industrial programming languages. Interestingly Esperanto even has some native (or first) speakers (Lindstedt, 2006). It should also be noted that there is at least one programming language that is a subset of a natural language. That language is – quite unsurprisingly – English. Inform 7 is a (highly domain-specific) programming language for creating interactive fiction using a natural language syntax. Inform 7 draws on ideas from linguistics and literate programming and is used in literary writing, games development and education.[7]

It is fairly well-known that fluent speakers of multiple natural languages can 'pick up' or acquire additional languages with an ease that seems much greater than that of learning one's first non-native language. Many programmers would say the same for programming languages. Portnoff (2018) makes a case that acquiring a second or subsequent programming language is even easier than it is for natural languages as "they all implement the same set of control and data mechanisms in very similar ways, the task of learning a second programming language for those with in-depth knowledge of a first programming language is more like learning a dialect than an entirely new language" (2018, p. 39). Arguably this makes learning one's first programming language as easily as possible extremely important as it can be seen as the main key to acquiring other languages, a trait common amongst, and very advantageous for, professional software developers.

2.1. English and Programming Languages

In general non-native English speakers program and learn in English (in as much as one can program in English), as almost all programming languages are designed using the English language as a base (Veerasamy & Shillabeer, 2014). Additionally, most sources of documentation are in English (Li & Prasad, 2005) as are most secondary sources of information such as Stack Overflow. Attempts to develop programming languages using natural languages other than English have been few, and have not gained popularity or use at university level teaching (Veerasamy & Shillabeer, 2014). However, many non-native English speakers, despite using languages that have English keywords choose to use their

[6] http://www.ppig.org/news/2006-06-01/linguistics-and-programming-languages
[7] http://inform7.com/about/

native language for comments, variable, method and function names.[8]

It is (probably) unlikely that a programming language will ever be created that is equivalent to a natural human language such as English, but being able to construct a computer program with a natural language would be obviously advantageous. Natural language programming, where a high-level programming language is either bypassed or constructed automatically from the input of natural language expressions has been researched for many years but is not currently near a state of useful widespread reality. For a review of such systems, see (Pulido-Prieto & Juárez-Martínez, 2017). However, imagine if perfect natural (English) language programming was achievable today. We can take this to be an extreme case along a continuum where at the opposite end programming languages have a syntax that is completely random. It is quite possible that the programmer's knowledge of English, or any other natural languages for that matter, would be of little use and therefore it is possible that non-native and native English speakers would be on an equal footing. Going back to a perfect (again English) natural language programming reality, it is not hard to imagine that if one can't speak English they have no chance whatsoever in constructing a program. The current situation of high-level languages designed in large part by English language speakers, with English keywords, and English resources, would put non-native English speakers at a disadvantage, but one between these two extremes. Figure 1 depicts this continuum and the hypothetical but plausible difficulty gap between native and non-native learners.

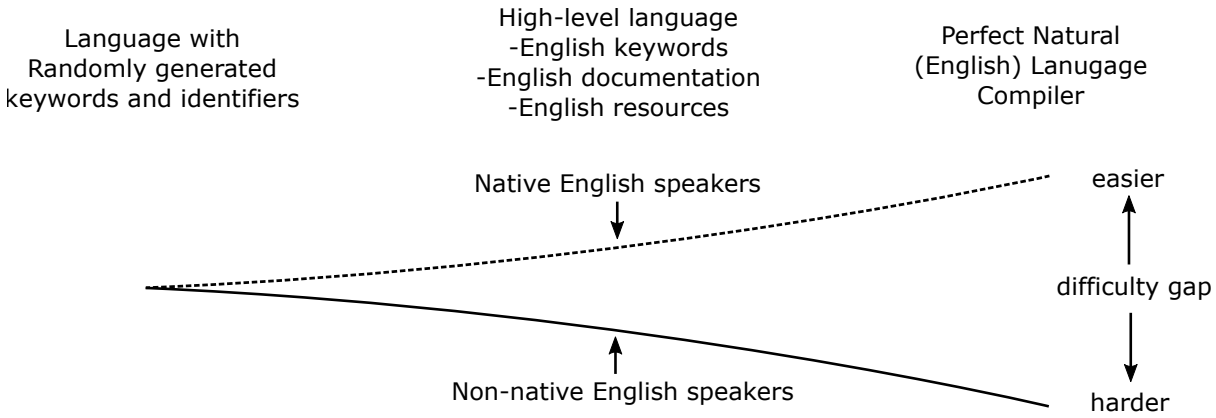

Figure 1 – A hypothetical difficulty gap between native and non-native English speakers grows as one progresses from a language with randomly generated keywords and identifiers through a typical (English) high-level language, through to a perfect (English) natural language compiler

How large this difficulty gap is, and what factors affect it is not well studied. Some influencing factors such as the computer language being used would obviously play a role but we do not currently have definitive answers for seemingly simple questions like: What is more difficult for non-native English speakers to learn, Java or C? It is worth noting that such questions are reasonable because the differences between computer languages, and therefore their differing relationships with natural languages can be substantive; for instance one of the biggest differences between object-oriented and non-object-oriented programming is the possibility to identify the actor of an action using purely syntactic means[6]. There is some recent work that sheds some light on this that we discuss in Section 4.1.

Our lack of knowledge in this area is at least in part because educators are far from agreement on what programming language is best for learning (and for whom), although this debate seems to have gained less attention in recent decades (Becker & Quille, 2019). It is possible that the debate of what language(s) are best for teaching will continue, perhaps indefinitely. However it is worth noting that there is an obvious reason that introductory programming courses rarely use low-level languages like assembly – it is accepted that it is too difficult and perhaps not that useful. Instead we use high-level languages that are by definition more like natural language. This observation alone provides sufficient

[8] https://softwareengineering.stackexchange.com/questions/1483/do-people-in-non-english-speaking-countries-code-in-english

```
poiblí aicme DiaDuitDomhan{
    poiblí statach folús príomh(Sreang[] args){
            // Priontáil "Dia duit, Domhan" go dtí an fuinneog teirminéal
            Córas.amach.priontáilln("Dia duit, Domhan");
    }
}
```

Listing 1 – How HelloWorld.java might look with Irish keywords and identifiers

motivation to explore the relationship between programming and natural languages and specifically, how this impacts the way that programming languages are learned.

2.2. An Example

Although to a native English speaker it may seem a somewhat trivial matter to have to deal with non-English keywords and identifiers, consider the fact that in Java the "Hello World" program – traditionally the first program a novice programmer writes – is almost entirely made up of keywords and identifiers. Listing 1 shows what a HelloWorld.java program might look like with Irish language keywords and identifiers, and a comment which is also in Irish.

Unless one reads Irish, it is probably not apparent at all what this program is, or what it would do. It is not unreasonable to assume that to someone who doesn't know English, or who has a limited grasp of English, would have similar difficulties with the traditional 'English' version of the Hello World program.

Anecdotally, the author has had conversations with a Greek colleague who has pointed that ironically, to use LaTeX, native Greek speakers have to learn the English names of Greek symbols such as delta (Δ) in order to typeset documents requiring these symbols. Although this may seem trivial, it is a real example of an often overlooked barrier that non-native English speakers can face. It is likely that there are many more examples such as this.

3. Programming Languages: Theories and the Human Brain

It might seem counter-intuitive that the small syntactic footprints of programming languages, with their relatively simple and compact grammars, would translate into a lengthy and involved learning process (Portnoff, 2018). Yet, citing (McCracken et al., 2001; Soloway, Bonar, & Ehrlich, 1983; Tew & Guzdial, 2011) amongst others, Scott and Ghinea (2013, p. 1) concluded that "despite considerable research into programming instruction since the inception of Computer Science as an academic discipline, many learners have not acquired the desired level of competency".

Evidence on the lack of theoretical approaches to teaching computer programming can be found in a recent review of 5,056 introductory programming papers from the period 2003-2017 – which eventually cited 735 papers – stating: "there are relatively few papers on theories of learning, with no obvious trends across the period of our review" (Luxton-Reilly et al., 2018, p. 66). This paper also provided supporting evidence that many papers on teaching programming don't have a theoretical foundation. Only 19 of the papers examined were coded as theory-related and the majority of these dealt with "learning styles" which have been largely discredited as "pseudoscience, myths, and outright lies" (Kirschner, 2017, p. 171).

Portnoff (2018, p. 38) also supported this view, stating:

> CS educators, however, currently operate with no evidence-based cognitive model for how students learn to program. When partial models have been invoked, they have generally presupposed the involvement of psychological constructs – such as that "cognitive loads" are lowered with drag-and-drop programming interfaces like Scratch or Alice – without having done research (i.e., taking experimental measurements) to corroborate such assumptions.

These are profound observations and insights when one considers the fact that learning programming necessarily involves learning a new (written, artificial) language, and the process of teaching and learning new natural languages has been well studied. Even 'language-agnostic' introductory programming courses that use pseudocode in essence require the learner to acquire a new language, and pseudocode has been argued as an unsuitable choice for assessment (Cutts, Connor, Michaelson, & Donaldson, 2014).

Bypassing any intermediate theory, a team consisting of a psychologist, a neurobiologist, a linguist, as well as computer scientists and software engineers, went straight to the source of programming learning – the human brain – conducting a controlled study on brain function of 17 participants using functional magnetic resonance imaging (fMRI)[9] while they were comprehending short source-code snippets which they contrasted with locating syntax errors (Siegmund et al., 2014). They found a clear, distinct activation pattern of five brain regions, which are related to working memory, attention, and language processing. The authors justly note that "Understanding program comprehension is not limited to theory building, but can have real downstream effects in improving education, training, and the design and evaluation of tools and languages for programmers" (2014, p. 378).

Writing about this study, Portnoff (2018, p. 36) reported: "The programmers in the study recruited parts of the brain typically associated with language processing and verbal oriented processing (ventral lateral prefrontal cortex). At least for the simple code snippets presented, programmers could use existing language regions of the brain to understand code without requiring more complex mental models to be constructed and manipulated."

What this means for how novices learn to program remains to be seen. It should be noted that fMRI studies such as this do have some threats to their validity – see (Siegmund et al., 2014, p. 385). If such anatomical studies prove to be robust it is likely that new theory will need to be developed, and new experiments carried out to test them, in order to inform the practice of teaching programming most effectively for both native and non-native English speakers.

These results do support the work of Portnoff (2018) who as part of a MSc thesis (Portnoff, 2016), as well as in his practice, argues that implicit (natural) language learning strategies are effective for teaching programming languages to novices. Portnoff found that applying foreign (natural) language pedagogies in programming instruction lead to a dramatic reduction in syntax issues with his students (Portnoff, 2018). He argues that the current prescriptive model of programming language instruction is at odds with the implicit way that native, and second (natural) languages are acquired.

4. Barriers to Programming faced by Non-native English Speakers

Guo (2018) pointed out that non-native English speakers face a range of well-known challenges in English-language classrooms in a wide range of disciplines including math, science, engineering, medicine, and the humanities. These ranged from cognitive to affective to social. Guo also pointed out that often these challenges, such as needing to mentally translate concepts into one's native language – especially in real time while listening to a lecture – increases extraneous cognitive load and decreases comprehension. Even difficulties with formulating verbal questions, and anxiety about a lack of English fluency makes these students less likely to ask clarifying questions. Bouvier et al. (2016) noted that the contextual background of a problem can also impact cognitive load (regardless of the native language of the student) and that computer science has unique characteristics compared to other disciplines, with the consequence that results from other disciplines may not apply to computer science, thus requiring investigation specifically within computer science. It should also be noted that different languages of instruction can hinder conducting and replicating research into these questions, further hampering progress in the area (Zingaro et al., 2018).

[9] fMRI (Functional Magnetic Resonance Imaging) measures brain activity by detecting changes associated with blood flow. This technique relies on the fact that cerebral blood flow and neuronal activation are coupled. When an area of the brain is in use, blood flow to that region also increases.

Given these well-accepted issues faced by non-native English speakers across many disciplines, and within computer science, it is not unrealistic to hypothesise that these learners may face barriers specific to learning to program. To investigate this, Guo (2018) conducted a survey of 840 responses from programmers spanning 86 countries and 74 native languages, identifying several barriers faced by non-native English speakers. Guo found that these programmers faced barriers with: reading instructional materials; technical communication (listening and speaking); reading and writing code; and simultaneously learning English and programming.

These respondents also expressed a desire for instructional materials to use simplified English without culturally-specific slang, more use of visuals and multimedia, more use of code examples that are culturally agnostic, and the incorporation of inline dictionaries. Additionally, some respondents reported that programming actually served as a motivating context for them to learn English better and helped clarify their logical thinking about natural languages, which provides further support for researching these barriers and how to help students overcome them.

Similarly, but perhaps counterintuitively, Li and Prasad (2005) found that native English speakers preferred examples and practice much more than non-native English speakers, and that non-native English speakers preferred lectures more than native English speakers. They found this to be consistent with their observations, but felt that this was possibly more of a cultural issue than a language issue. Further research needs to be carried out to explore these issues.

Supporting the theory that non-native English speakers face more severe barriers than native English speakers when programming, Dasgupta and Hill (2017) found that novice users who code with their programming language keywords and environment localised into their home countries' primary language (German, Italian, Norwegian Bokmål, Portuguese, and Brazilian Portuguese) demonstrated new programming concepts at a faster rate than users from the same countries whose interface was in the default of English. In developing Spoken Java, a semantically identical variant of Java that is easier to say out loud, Begel and Graham (2005) found several differences between how native and non-native English speakers vocally express in code. An example is the use of Prosody (volume, timbre, pitch, and pauses). They found that the semantic use of prosody was limited mostly to native English speakers – many non-native English speakers who speak English typically use the prosody of their native language, in which pauses, in particular, do not hold the same meaning. This affected how spoken Java was interpreted for instance when dealing with brackets and punctuation.

In the following subsections we explore two very different classes of barriers faced by non-native English speakers when learning to program. It should be stressed that these barriers are present for both native and non-native English speakers, but they might affect these groups differently. These two classes are only two of potentially many more. First we look into a core aspect of programming – dealing with keywords and syntax. We then look into other aspects of the code base – error messages and code comments.

4.1. Syntax, Keywords and Reference Errors

Miller and Settle (2019) explained how novice programmer reference errors are consistent with the use of metonymy, a form of figurative expression in human communication where the name of an attribute is substituted for the name of something closely associated with that attribute; for example 'suit' being used as a substitute for 'business executive'. Miller (2016) provided three possible knowledge sources for why novice programmers produce reference-point errors that are consistent with the use of metonymy. Some of these knowledge sources may differ between native and non-native English speakers, implying that these groups of students may experience the complex relationship between natural and programming languages differently. One of these sources involves misconceptions about notional machines which brings up a question on if native and non-native English speakers may form different models of notional machines. Miller and Settle (2019, p. 2) note that "In contrast to the relative ease with which humans comprehend figurative language such as metonymy, it presents difficulties with human-to-machine communication, particularly in the domain of programming". They also showed that

the presentation of examples can affect the construction of references in student solutions. They suggest that reference-point errors may be the result of well-practiced habits of communication rather than misconceptions of the task or what the computer can do. As the habits of communication between native and non-native English speakers differ to varying extents, it is most likely that these two groups of students will face different difficulties, or difficulties of varying severity, when it comes to constructing and interpreting programs that contain references influenced by this mechanism.

An examination of keywords in high-level programming languages showed that they are also formed using mechanisms analogous to those observed in the English – for instance, the choice of keywords by language designers is similar to neologism formation in English (Eastman, 1982). A neologism is a new word; it may be either a newly created word or an existing word whose meaning has changed (1982). This process is also related to the choice of identifier names by the programmer. Eastman also noted a conspicuous exception; the use of mirror words such as `fi` to close an `if` statement. This might not be as trivial as it sounds. Mirror keywords can evoke strong reactions. Writing about `fi`, Don Knuth stated: "I don't really like the looks of `fi` at the moment; but it is short, performs a useful function, and connotes finality, so I'm confidently hoping that I'll get used to it" (1974, p. 266). In the same paper he stated that Alan Perlis "has remarked that `fi` is a perfect example of a cryptic notation that can make programming unnecessarily complicated for beginners" (1974, p. 266). These reactions are based on the fact that those doing the reacting correctly recognised that they are in fact mirror words. A non-native English speaker might miss this. Therefore it is possible that in some cases, native and non-native English speakers may react differently (and possibly strongly) to keywords. If these groups react differently, it would not be surprising if they find learning and using them to be different experiences. If how these groups use a language differ, it is likely that how they would best learn that language would differ also.

Eastman (1982) put forward a good reason why mirror words could seen as nonsense – one could almost take them as being random. Interestingly, Stefik and Siebert (2013) carried out four experiments on (largely native English speaking) novice programmer accuracy rates using six programming languages: Ruby, Java, Perl, Python, Randomo, and Quorum. Randomo was designed by randomly choosing keywords from the ASCII table. They found that that Perl and Java – languages using a more traditional C-style syntax – did not afford accuracy rates significantly higher than Randomo, a language with randomly generated 'gibberish' keywords. However they found that Quorum, Python and Ruby – languages which do deviate from a traditional C-style syntax – did. One of the main conclusions drawn by Stefik and Siebert (2013) was: syntax does matter to novices and accuracy rates vary by language. Given this it is quite probable that the experience of non-native English speakers would also vary according to language. The question is, how would their experience differ? It is interesting that the results for Randomo were not worse than some well-established languages. This could lead to a hypothesis that at least for these languages, the experience of non-native English speakers may be similar to native speakers. Clearly more work needs to be carried out in this area.

Keyword and identifier names also share similarity to natural language words in that they are often compound. Additionally, abbreviations (and acronyms) are not uncommon in both natural language and programming keywords and identifiers. Keywords made up of parts of existing words can be regarded as blends – something between compounds and acronyms. Suffixes and prefixes are also occasionally used. Guo (2018) provided references that Non-native speakers report struggling to decipher the meanings of code identifiers, especially when they are abbreviated; for instance the C function `getch()` stands for "get character". Liblit, Begel, and Sweetser (2006) found that programmers choose and use names (for programming constructs) in regular, systematic ways that reflect deep cognitive and linguistic influences. Blackwell (2006) found several categories of vocabulary used in Java documentation revealing extremely complex terminology with similarly complex underlying concepts. These findings also indicate that non-native English speakers may face substantial difficulty in navigating code and documentation.

4.2. Programming Messages and Code Comments

A specific facet of the programming experience that can be affected by natural language ability is dealing with programming messages – error, warning, or other messages resulting from errors with code that result in what are commonly called 'compiler error messages'. These messages have been shown to be a barrier to learning for students (Becker et al., 2019), including both native and non-native English speakers (Becker et al., 2018; Ko, Myers, & Aung, 2004). Arguably these messages, in an 'English' programming language, should be comprised of English text that is comprehensible to English speakers, and for native English learners should be easy to interpret, allowing for effective error resolution. An effective message therefore, by definition, should be presented in plain English as much as possible. It is simple to conclude that if native English speakers have trouble with these messages, non-native speakers would have at least as much trouble as native speakers, and in most cases, more. Ko et al. (2004) noted that attempts to translate APIs and error messages have faced a lack of adoption since programmers cannot as easily search for online help using the localised terms. It should also be noted that some of the difficulties with error messages such as 'cascading' error messages (Becker et al., 2018) likely affect native and non-native English speakers similarly. However, other difficulties likely affect these groups differently.

Programming messages are a part of the programmer-facing code base on the output side. They are intended to be read and interpreted by humans. Similarly, another aspect of programming, but on the input side, are code comments. Like error messages, code comments are an essential part of programming, and somewhat differently to writing code itself, are primarily intended to be read and interpreted by other humans. Comments are written largely in natural language, and therefore require a high degree of fluency in the language being used. The author is unaware of any studies that investigate how non-native English speakers programming in 'English' programming languages write comments. The most related work found was Stefik and Siebert (2013) who reported that when creating single-line comments, non-programmers rated the English words `note` and `comment` highly. Interestingly, non-programmers rated the traditional C-style single line comment denotation `\\` approximately the same as `note` and `comment`. Similar to the discussion in Section 4.1, one could hypothesise that using `note` and `comment` would be more disadvantageous for non-native English speakers. Interestingly though, the results for `\\` could lead to a hypothesis that there are ways of denoting comments that may be similar in usefulness to native and non-native English speakers. Again, it is clear that more work is needed on this front.

5. Questions

In this section we enumerate some of the questions that arise from the topics discussed in this paper and can be used for the basis of future work in the area.

1. How similar are the processes of learning programming languages and natural languages?
   (a) How is learning a programming language different for native and non-native English speakers?
   (b) It is obvious that natural languages are much more difficult to parse by computational means than programming languages. This implies that the human parsing mechanism works quite differently to computational parsing. Does that mean that humans (regardless of natural language) experience a 'natural' difficulty in parsing programming languages because of biology?
2. Do non-native English speakers have more difficulty programming specific languages compared to native English speakers?
   (a) If so, for what languages is the experience for non-native English speakers more similar to that of native English speakers?

3. What techniques can be borrowed from natural language acquisition that would improve programming language acquisition?

    (a) Would non-native English speakers benefit from these techniques in the same way as native English speakers?

4. Apart from the programming language itself, how do different mediums of instruction, and different tools such as IDEs impact how non-native English speakers learn to program?

5. How do specific facets of the programming experience such as keywords, syntax, comments and error messages affect non-native English speakers?

6. Future Work

There have been attempts to create programming languages with non-English keywords, but none have been widely adopted, and attempts to translate APIs and error messages have faced a similar lack of adoption (Guo, 2018). There are also non-English programming environments for languages such as Arabic (Al-Salman, 1996), but these have not been adopted widely (Veerasamy & Shillabeer, 2014).

Inspired by Dasgupta and Hill (2017), the author has piloted a study with approximately 120 non-native English speakers using an online IDE. These students are all native Chinese (Mandarin) speakers enrolled in an introductory programming course as part of Computing/Engineering degrees. The next phase involves a study where similar students will be provided the same IDE with both English and Chinese (Mandarin) interfaces. The IDE currently allows programs to be written in C, Java, and Prolog. The planned research questions are:

1. Do non-native English speakers receiving programming instruction in English prefer to use an IDE in their native language when given the choice?

2. What barriers does the IDE present to non-native English speakers who are learning to program?

3. Is there a correlation between performance and IDE language for non-native English speakers?

7. Conclusion

Learning to program is fraught with challenges and there have been numerous studies over several decades exploring the barriers that novices face when learning to program. However, how non-native English speakers learn to program, and how this differs from native English speakers, is an understudied area.

This paper set out several of the high level issues that non-native English speakers may face when learning to program. There is mounting evidence that there are commonalities between how natural and programming languages are learned, but very little has been carried out on how this would affect non-native English speakers.

It is probable that the barriers that non-native English speakers experience when learning to program are different to those that native English speakers face. Many of these barriers may affect both native and non-native English speakers, but could affect non-native English speakers to a greater extent. Most likely, there are barriers to learning programming that are faced by non-native English speakers that native English speakers do not face.

This paper presented an overview of some of the challenges that non-native English speakers face when learning to program. It also presented several unanswered questions that may lead to future research that may help these students overcome such challenges. It also presented the author's planned work in the area of how the language of the programming environment affects non-native English speakers. It should be noted that this is not a comprehensive literature review, but the author has been collecting papers on the relationship between natural and programming languages and on how non-native English

speakers learn to program for years and this paper cites about half of that collection. The interested reader is guided to the following as entry points for further reading Guo (2018); Pal (2016); Portnoff (2018, 2016).

Particularly in a time when broadening participation in computing is seen as a primary objective, the community can no longer afford to overlook the unique barriers faced by non-native English speakers who want to learn to program.

8. Acknowledgements

The author would like to thank Dónal and Mairéad Holohan for their help with the 'Irish Java' translation.

9. References

Al-Salman, A. S. (1996). *An arabic programming environment* (Unpublished doctoral dissertation).

Becker, B. A. (2015). An exploration of the effects of enhanced compiler error messages for computer programming novices. *Thesis* (November). Retrieved from https://arrow.dit.ie/ltcdis/35/ (https://www.brettbecker.com/publications)

Becker, B. A., Denny, P., Pettit, R., Bouchard, D., Bouvier, D. J., Harrington, B., ... Prather, J. (2019). Unexpected tokens: A review of programming error messages and design guidelines for the future. In *Proceedings of the 2019 acm conference on innovation and technology in computer science education* (p. 253–254). New York, NY, USA: Association for Computing Machinery. Retrieved from https://doi.org/10.1145/3304221.3325539 doi: 10.1145/3304221.3325539

Becker, B. A., Murray, C., Tao, T., Song, C., McCartney, R., & Sanders, K. (2018). Fix the first, ignore the rest: Dealing with multiple compiler error messages. In *Proceedings of the 49th ACM Technical Symposium on Computer Science Education - SIGCSE '18* (pp. 634–639). Baltimore, MD, USA: ACM. Retrieved from http://dl.acm.org/citation.cfm?doid=3159450.3159453 doi: 10.1145/3159450.3159453

Becker, B. A., & Quille, K. (2019). 50 Years of CS1 at SIGCSE: A review of the evolution of introductory programming education research. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education - SIGCSE '19* (pp. 338–344). Minneapolis, MN: ACM. Retrieved from http://dl.acm.org/citation.cfm?doid=3287324.3287432 doi: 10.1145/3287324.3287432

Begel, A., & Graham, S. (2005). Spoken programs. In *2005 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC'05)* (pp. 99–106). IEEE. Retrieved from http://ieeexplore.ieee.org/document/1509493/ doi: 10.1109/VLHCC.2005.58

Blackwell, A. F. (2006). Metaphors we program by: Space, action and society in Java. In *18th Workshop of the Psychology of Programming Interest Group - PPIG '06*. Retrieved from http://www.ppig.org/library/paper/metaphors-we-program-space-action-and-society-java

Bouvier, D., Lovellette, E., Matta, J., Alshaigy, B., Becker, B. A., Craig, M., ... Zarb, M. (2016). Novice programmers and the problem description effect. In *Proceedings of the 2016 ITiCSE Working Group Reports* (pp. 103–118). New York, NY, USA: ACM. Retrieved from http://doi.acm.org/10.1145/3024906.3024912 doi: 10.1145/3024906.3024912

Crestani, M., & Sperber, M. (2010). Experience report: growing programming languages for beginning students. *ACM Sigplan Notices*, *45*, 229–234. Retrieved from http://dl.acm.org/citation.cfm?id=1863576

Cutts, Q., Connor, R., Michaelson, G., & Donaldson, P. (2014). Code or (not code) – Separating formal and natural language in CS education. In *Proceedings of the 9th Workshop in Primary and Secondary Computing Education - WiPSCE '14* (pp. 20–28). New York, New York, USA: ACM Press. Retrieved from http://dl.acm.org/citation.cfm?doid=2670757.2670780 doi: 10.1145/2670757.2670780

Dasgupta, S., & Hill, B. M. (2017). Learning to code in localized programming languages. In *Pro-*

ceedings of the Fourth (2017) ACM Conference on Learning @ Scale - L@S '17 (pp. 33–39). Cambridge, MA, USA: ACM Press. Retrieved from http://dl.acm.org/citation.cfm?doid=3051457.3051464 doi: 10.1145/3051457.3051464

Eastman, C. M. (1982). A comment on English neologisms and programming language keywords. *Communications of the ACM*, *25*(12), 938–940. doi: 10.1145/358728.358756

Guo, P. J. (2018). Non-native English speakers learning computer programming. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems - CHI '18* (pp. 1–14). New York, New York, USA: ACM Press. Retrieved from http://dl.acm.org/citation.cfm?doid=3173574.3173970 doi: 10.1145/3173574.3173970

Kirschner, P. A. (2017). Stop propagating the learning styles myth. *Computers and Education*, *106*, 166–171. Retrieved from http://dx.doi.org/10.1016/j.compedu.2016.12.006 doi: 10.1016/j.compedu.2016.12.006

Knuth, D. E. (1974, dec). Structured programming with go to statements. *ACM Computing Surveys*, *6*(4), 261–301. Retrieved from http://portal.acm.org/citation.cfm?doid=356635.356640 doi: 10.1145/356635.356640

Ko, A. J., Myers, B. A., & Aung, H. H. (2004). Six learning barriers in end-user programming systems. *Proceedings - 2004 IEEE Symposium on Visual Languages and Human Centric Computing*, 199–206. Retrieved from http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1372321 doi: 10.1109/VLHCC.2004.47

Li, X., & Prasad, C. (2005). Effectively teaching coding standards in programming. In *Proceedings of the 6th Conference on Information Technology Education - SIGITE '05* (p. 239). New York, New York, USA: ACM Press. Retrieved from http://portal.acm.org/citation.cfm?doid=1095714.1095770 doi: 10.1145/1095714.1095770

Liblit, B., Begel, A., & Sweetser, E. (2006). Cognitive perspectives on the role of naming in computer programs. In *18th Workshop of the Psychology of Programming Interest Group - PPIG '06*. Retrieved from http://www.ppig.org/library/paper/cognitive-perspectives-role-naming-computer-programs

Lindstedt, J. (2006). Native Esperanto as a test case for natural language. *SKY Journal of Linguistics*, *19*(SUPPL), 47–55. Retrieved from http://www.linguistics.fi/julkaisut/SKY2006_1/1FK60.1.5.LINDSTEDT.pdf

Luxton-Reilly, A., Sheard, J., Szabo, C., Simon, Albluwi, I., Becker, B. A., ... Scott, M. J. (2018). Introductory programming: a systematic literature review. In *Proceedings Companion of the 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education - ITiCSE 2018 Companion* (pp. 55–106). New York, New York, USA: ACM Press. Retrieved from http://dl.acm.org/citation.cfm?doid=3293881.3295779 doi: 10.1145/3293881.3295779

McCracken, M., Wilusz, T., Almstrum, V., Diaz, D., Guzdial, M., Hagan, D., ... Utting, I. (2001). A multi-national, multi-institutional study of assessment of programming skills of first-year CS students. In *Working Group Reports From ITiCSE on Innovation and Technology in Computer Science Education - ITiCSE-WGR '01* (Vol. 33, p. 125). New York, New York, USA: ACM Press. Retrieved from http://portal.acm.org/citation.cfm?doid=572133.572137 doi: 10.1145/572133.572137

Miller, C. S. (2016). Human language and its role in reference-point errors. In *27th Workshop of the Psychology of Programming Interest Group - PPIG '16*. Retrieved from http://www.ppig.org/library/paper/human-language-and-its-role-reference-point-errors

Miller, C. S., & Settle, A. (2019). Learning to get literal: Investigating reference-point difficulties in novice programming. *ACM Transactions on Computing Education*, *19*(3), 1–17. Retrieved from http://dl.acm.org/citation.cfm?doid=3308443.3313291 doi: 10.1145/3313291

Naur, P. (1975). Programming languages, natural languages, and mathematics. *Communications of the*

ACM, *18*(12), 676–683. doi: 10.1145/361227.361229

Pal, Y. (2016). *A framework for scaffolding to teach programming to vernacular medium learners* (Unpublished doctoral dissertation).

Portnoff, S. R. (2016). The case for using foreign language pedagogies in introductory computer programming instruction. *ProQuest Dissertations and Theses*. Retrieved from https://search.proquest.com/openview/8f4a5a498c2ba52b27787cc79041b955/1?pq-origsite=gscholar&cbl=18750&diss=y

Portnoff, S. R. (2018). The introductory computer programming course is first and foremost a language course. *ACM Inroads*, *9*(2), 34–52. Retrieved from http://dl.acm.org/citation.cfm?doid=3211407.3152433 doi: 10.1145/3152433

Pulido-Prieto, O., & Juárez-Martínez, U. (2017). A survey of naturalistic programming technologies. *ACM Computing Surveys*, *50*(5), 1–35. doi: 10.1145/3109481

Scott, M. J., & Ghinea, G. (2013). Educating programmers: A reflection on barriers to deliberate practice. In *Proceedings of the HEA STEM Learning and Teaching Conference.* Birmingham, UK: Hihgher Education Academy. Retrieved from http://repository.falmouth.ac.uk/1650/

Siegmund, J., Kästner, C., Apel, S., Parnin, C., Bethmann, A., Leich, T., ... Brechmann, A. (2014). Understanding understanding source code with functional magnetic resonance imaging. In *Proceedings of the 36th International Conference on Software Engineering - ICSE 2014* (pp. 378–389). New York, New York, USA: ACM Press. Retrieved from http://dl.acm.org/citation.cfm?doid=2568225.2568252 doi: 10.1145/2568225.2568252

Soloway, E., Bonar, J., & Ehrlich, K. (1983). Cognitive strategies and looping constructs: an empirical study. *Communications of the ACM*, *26*(11), 853–860. doi: 10.1145/182.358436

Stefik, A., & Siebert, S. (2013). An empirical investigation into programming language syntax. *ACM Transactions on Computing Education*, *13*(4), 1–40. Retrieved from http://dl.acm.org/citation.cfm?doid=2543488.2534973 doi: 10.1145/2534973

Tenenberg, J., & Kolikant, Y. B. D. (2014). Computer programs, dialogicality, and intentionality. In *Proceedings of the 10th International Computing Education Research Conference - ICER '14* (pp. 99–106). New York, New York, USA: ACM Press. Retrieved from http://dl.acm.org/citation.cfm?doid=2632320.2632351 doi: 10.1145/2632320.2632351

Tew, A. E., & Guzdial, M. (2011). The FCS1: A language independent assessment of CS1 knowledge. In *Proceedings of the 42nd ACM Technical Symposium on Computer science Education - SIGCSE '11* (p. 111). New York, New York, USA: ACM Press. Retrieved from http://portal.acm.org/citation.cfm?doid=1953163.1953200 doi: 10.1145/1953163.1953200

Veerasamy, A. K., & Shillabeer, A. (2014). Teaching English based programming courses to English language learners/non-native speakers of English. *International Proceedings of Economics Development and Research*, *70*(4), 17–22. Retrieved from http://www.ipedr.com/vol70/004-ICEMI2014_H00006.pdf

Zingaro, D., Craig, M., Porter, L., Becker, B. A., Cao, Y., Conrad, P., ... Thota, N. (2018). Achievement goals in cs1: Replication and extension. In *Proceedings of the 49th ACM Technical Symposium on Computer Science Education - SIGCSE '18* (pp. 687–692). New York, NY, USA: ACM. Retrieved from http://doi.acm.org/10.1145/3159450.3159452 doi: 10.1145/3159450.3159452