The Elephant in the Room

Andreas Stefik
University of Nevada, Las Vegas
{stefika@gmail.com}

Abstract
In recent years, the programming language design community has engaged in rigorous debate on the role of empirical evidence in the design of general purpose programming languages. Some scholars contend that the language community has fundamentally failed to embrace the use of empirical evidence in evaluating the impact of language design on the people that use such technologies. Other scholars contend that the programming language wars, as they are colloquially called, are too complex and cannot be studied scientifically. While the discussion will persist, the computer science education community is also deeply fractured in regards to its use of programming languages. Educators at universities appear to choose whatever technologies they deem appropriate, while educators in K-12 follow along. In this white paper, I discuss the elephant in the room, arguing that the programming language wars is having a detrimental impact on the educational community and the world at large. Serious discussions about the future directions of computer science education must consider the impact it has and consider strategies to alleviate the most crucial problems.

Categories and Subject Descriptors D.3 [Programming Languages]; H.1.2 [Information Systems]: User/Machine Systems — Software Psychology

General Terms Design, Experimentation, Human Factors, Standardization

Keywords Empirical Evidence, Programming Languages

1. Introduction
Programming language design is a complex and difficult field of study. Few reasonable computer scientists would question that language design must take into account a host of complex and sometimes competing factors. Papers on language design are often accompanied by proofs that a particular feature will work, performance studies that afford clues on the efficiency of a system, or claims on how a system might be used by educators, students, or professionals. The right mix of syntax, semantics, design features, or accompanying tools (e.g., the whyline [14]) that best afford human productivity in general purpose programming languages is a matter of rigorous debate.

While recognizing that language design requires a deep understanding of a complex set of concerns, before discussing what I think are crucial research questions for the CS education community, it behooves us to consider the recent debate in the major language conferences. In sum, several authors have made very specific criticisms regarding the state of the modern language design literature. Namely, Hanenberg contends that the programming language design community is so evidence-poor that it borders on a faith-based discipline [9]. Markstrum claims that even when human factors are considered by the language community, claims are built on a foundation of anecdotes—a form of evidence that would be considered suspect in many scientific disciplines [15].

Tichy [23] made similar arguments more than a decade ago, claiming that this non-evidence based foundation in regards to human factors decisions with language design has persisted for decades. If Hanenberg, Markstrum, and Tichy are correct, the fundamental issue with the language wars may not be its complexity, but a scientific community unwilling to spend the months, or years, it takes to conduct serious investigations into the issue. Regardless of the reasons behind ignoring the language wars, the claims made by these authors are serious allegations against the scientific validity of the academic literature on language design—claims that, if true, should not be dismissed lightly by serious scholars.

In addition to scholarly claims against the scientific validity of the academic literature in regards to human factors concerns in language design, there are at least three compelling reasons the computer science education community should consider the impact of the programming language wars, namely: 1) the language wars are poorly understood, 2) research shows language design decisions significantly impact humans (especially students), and 3) the language wars have significant social implications for our world. Through-
out the rest of this paper, I discuss these three major issues, give research questions I think are relevant, and discuss a limited and carefully chosen selection of related work, focusing most on those papers that include formal randomized controlled trials or similarly sophisticated experiments.

2. The language wars are poorly studied

Hanenberg, Markstrum, and Tichy have made claims that language design has not historically been evidence based in regard to human factors. While no systematic investigation into the allegations have yet been published, I am currently working with a large team of scholars to document formally whether they are true. Thus, in this section, I provide preliminary findings from this work, analyzing two issues that lead us, in my view, toward research questions relevant for the future directions of CS education.

First, while I know of no systematic analysis of the types and kinds of programming languages used at colleges and universities across the United States, or the world, it may be enlightening to capture this information on a global scale. While an admittedly inadequate start, I began such an analysis by conducting a short survey of 39 colleges and universities in the mid-west, analyzing only what languages were used by these schools in the introductory course. Of these, approximately 9 general purpose programming languages were used, including C, C++, Alice (although debatably general purpose), VB, Python, Java, C#, COBOL, and ADA. These languages are very different.

Interestingly, in personal contact with professors teaching these courses, opinions toward the reasons varied considerably. Some individuals that responded to the informal poll appeared to believe that the language chosen was the “right” one. As should be clear, however, not everyone can be correct, unless it is reasonably proven that all programming languages are equally easy/hard to use. Given the evidence presented in Section 3, however, such a claim is almost certainly false. This leads us to our first research question, which is inspired by the sociological work of Meyerovich and Rabkin [17]:

| RQ1 | What causes academic institutions (e.g., K-12, Colleges, Universities) to adopt particular programming languages? |

Second, if we carefully analyze the academic literature on language design, there is reason to be suspicious of the community’s claims in regard to human factors. For example, a recent paper by Ko et al. demonstrated that, in the last 10 years at top software engineering conferences, only approximately 44 controlled experiments involving humans have been published [13] in the field of software engineering, a discipline that supposedly studies how humans develop software. In the language design discipline, preliminary analysis I am conducting shows controlled experiments with humans are exceedingly rare (e.g., in POPL, OOPSLA, ECOOP). With that said, two academic workshops are often cited as having papers, namely PPIG and Plateau, venues I will discuss next.

While many believe PPIG papers often study language design issues, there is reason for skepticism of this view. For example, if we look at work from predominately the years 1999 through 2011, only a small collection of papers analyzed human factors in the design of languages. Specifically, for this period, I cataloged 257 papers, of which I coded whether a paper, 1) conducted a study involving human users, 2) used quantitative evidence, or 3) used qualitative evidence. Then, I documented 4) whether and how a paper was relevant to the design of programming languages (e.g., functions, inheritance), and 5) approximately how many pages in the paper included empirical evidence (e.g., study descriptions, data, statistics, transcripts from interviews). While the data presented here is preliminary, initial results from PPIG show that approximately 27.5% of pages presented either empirical evidence or details on the methodology for documenting that data. However, while PPIG is focused on the psychology of programmers, I found that 75.0% of the papers coded from this workshop either did not include evidence or were not particularly relevant to the design of programming languages. Further, only 5.86% of papers both included evidence, of any kind or quality, and discussed the type of text-based languages most commonly used in a typical computer science educational sequence.

Further, given that the OOPSLA community has recently established a workshop called Plateau, which has similar goals to PPIG but is held in the United States, I further analyzed this venue. In this case, 68.6% of papers included some kind of empirical study involving humans and 28.1% of the total pages included some kind of empirical information. However, 48.6% of papers either included no evidence or were not related to general purpose programming languages. In this case, 31.4% of papers fit into categories that are most directly related to the design of general purpose programming languages. Unlike PPIG, which is a vibrant community, Plateau has been relatively inactive, with 8 papers published in its first year (2009), up to its peak of 12 (2011), then to 5 (2012). The 2013 version of Plateau was cancelled, giving zero papers for that year. Analysis on SIGCSE is ongoing.

Finally, although crucially, given that top conferences in language design have historically published little empirical evidence on human factors, given that PPIG and Plateau are workshops, their experiments are generally preliminary. For example, my team is currently evaluating all known experiments on human factors in language design using the U.S. Government’s “What Works Clearinghouse” standard for empirical studies [18]. Unfortunately, preliminary findings indicate that the studies generally fall into the “Does not Meet Standards” criterion. In effect, many experiments
are missing what most scientific disciplines consider to be crucial facets of experimental design, like having control groups. These problems with experimental design were very common in the medical sciences in the 18th and 19th centuries [11]. This problem seems to be especially exacerbated at PPIG, where authors have heavily used the Cognitive Dimensions of Notations Framework [8]. It appears that many authors have done so so informally as to be practically Laissez-Faire. As Tichy points out [23], other modern disciplines appear to have more rigorous control standards in academic journals than computer science does. In effect, in some disciplines (e.g., top journals in psychology and medicine), the use of empirical data, evidence, and checks and balances like control groups is simply not optional. This leads us to our second research question:

| RQ2 | Why is the academic community not using standard scientific evidence gathering procedures in regards to the language wars, in terms of both quantity and quality of scholarly reports? |

3. Language design decisions impact humans

While many computer science educators likely have an intuitive sense that different language designs can impact students, recent studies have increasingly shown that this impact is, perhaps, much larger than was once assumed. For example, Denny et al. conducted a recent empirical study showing that programming language syntax in Java remains a major barrier to students [5]. In an experiment involving 330 students, Denny et al. found that students in the top quartile submitted non-compiling source code approximately half the time, whereas those in the bottom quartile submitted non-compiling code 73% of the time. The broad message of Denny et al.’s work is that even excellent students in an introductory programming course experience syntax issues. Later work has shown that syntax errors vary in how difficult they are for novices to solve [6].

Similarly, in my own work, I have conducted randomized controlled trials with novices. Results show clear evidence that 1) syntax remains difficult to understand for novices, 2) that designing effective syntax is non-trivial, and 3) that languages, and language constructs, vary in their impact on novices. For example, some assume that “natural” or “plain English” statements help novices, but this view is only partially correct. For example, in the design of “if statements” the words “if then” would be plain English, but having the word “then” in the syntax decreases novice syntactic accuracy significantly. In contrast, using the word “repeat” increases novice syntax accuracy when writing loops, when compared to words like “for.” In short, syntax needs to be very carefully evaluated [20]. We must stop accepting papers that include only anecdotal claims in language design, even if the languages are new or novel [2]. This leads to the next research question:

| RQ3 | Given the formally documented difficulties novices have with syntax, which design decisions best afford novice success? |

However, some might believe that a study of general purpose programming languages (especially syntax) is not worthwhile, given innovations by those developing syntax directed editors [22], or more modern tools like Alice, Scratch, or perhaps end user programming systems. In this regard, it is crucial to recognize that visual tools do help novices initially (although not for blind users), especially in promoting transfer-of-learning from visual systems to text-based programming languages. This has been confirmed independently and using different methodologies with at least the tool ALVIS [10] and Alice [4]. However, visual tools are not a silver bullet. Garlick and Cankaya compared using Alice in an introductory course to a control group that started with pseudo code, finding that the Alice group had a statistically significant drop in grades [7]; a very different kind of study with a very different outcome from the one discussed by Dann et al. [4]. Regardless of the ongoing debate, despite decades of study on tools designed to bypass syntax, general purpose programming languages are still overwhelmingly used in the classroom and in industry. This leads us to a fourth research question:

| RQ3 | What would or should the impact of visual programming tools be on computer science courses beyond the first, given the continued overwhelming use of general purpose programming languages in practice? |

Further, some educators believe that languages with dynamic (or static) typing hold advantages for humans. While this debate has existed for some time, only recently have formal randomized controlled trials been conducted [12, 16]. Results from this work show that by around approximately year 3 of an academic pipeline, students have a statistically significant increase in the speed in which they can use an API, under a variety of conditions, using statically typed languages. However, studies on syntax show that novices have significant difficulty with type annotations, with the evidence suggesting that annotations inside of method declarations are the most difficult—precisely the location where more advanced students show benefits [20]. While this situation may seem unresolvable, in the programming language Quorum, results from all known type system studies have been combined, using a complex combination of inference and static typing, but this system has not been studied longitudinally to see if it is the optimal trade-off for students at all levels of the academic pipeline. Given this, we have a fifth research question:
Finally, while there are many features that need study in language design (e.g., infix vs prefix notations, the use of closures, competing syntax/semantics for inheritance), spelling out research questions for all of them could take a treatise far larger than is likely intended for these white papers. Given this, while I could choose many features, given its increasing use, concurrency is another area where students exhibit very large problems with competing language designs. For example, Lewandowski et al. showed that 97% of students could identify race conditions in simple concurrent code, with 71% able to provide English descriptions of a solution. However, Rossbach et al., showed that students programming using coarse or fine grain locks exhibited approximately 7 times more bugs in their code compared to those using transactional memory [19]. Together, these studies indicate that students may be able to grasp concurrency much easier than some believe, but only if the programming language represents the concepts sensibly. Many facets needs to be considered (e.g., syntax, a host of competing technologies), but one broad research question we could ask might be:

| RQ5 | Given that students show initial syntactic difficulties with type systems, but that they are benefited later, what type system design affords the highest productivity for the greatest number of people? |

5. Conclusion

No academic paper, especially not a short unpublished white paper, can begin to explore with rigor the types of deep problems that are apparent with the programming language wars. Like the mysteries in the medical sciences, the language wars are deeply complex. With that said, new languages, and especially web technologies, are invented at a dizzying pace and are being adopted by academics and industry alike. New features are added to programming languages (e.g., closures in JDK 8 and C++), regularly. Since students inevitably encounter the syntax, semantics, and full standard libraries of these languages, the features must either be ignored at our peril or placed into an often full pedagogical queue. Perhaps most disappointingly, these new technologies, have been seemingly created without the faintest shred of scientific evidence to support their human factors decisions. We need to tackle the programming language wars. We need to tackle the elephant in the room.

References


