



# Methodological Irregularities in Programming-Language Research

**Andreas Stefik**, University of Nevada, Las Vegas

**Stefan Hanenberg**, University of Duisburg-Essen

*Substantial industry and government investments in software are at risk due to changes in the underlying programming languages, despite the fact that such changes have no empirically verified benefits. One way to address this problem is to establish rigorous evidence standards like those in medicine and other sciences.*

**S**oftware plays an essential role in modern life and is a major economic driver. According to Gartner, software sales in 2013 topped \$407 billion, a nearly 5 percent increase over the previous year.<sup>1</sup>

Changes in the languages used to program software necessarily affect the entire industry. Developers must

implement new processes and policies, and users—whether enterprises, government agencies, or consumers—must adapt to the changes by learning new functions, conventions, and terminology.

Programming-language changes also impact education, where science, technology, engineering, and mathematics (STEM) programs such as Computer Science for All<sup>2</sup> in the US are receiving significantly increased government and corporate support. As languages change, curricula must be updated and teachers retrained. For example, Exploring Computer Science ([www.exploringcs.org](http://www.exploringcs.org)), which aims to democratize computer science in K-12 schools in the US, uses seven programming or markup languages: HTML, CSS, JavaScript, R, Flash, Scratch, and Lego Mindstorms.

Given the substantial economic investment in software and its importance to all aspects of society, one would expect the industry to use rigorous empirical methodologies to ascertain whether the benefits of eliminating or modifying programming languages or introducing new ones outweigh the disadvantages. As



## HISTORICAL EVIDENCE STANDARDS IN SCIENCE

Perhaps the first evidence standard in scientific experimentation was the blind trial, introduced as a research protocol in the late eighteenth century to eliminate the potential impact of bias, preconceived opinions, or imagination. Tasked by Louis XVI of France with assessing Dr. Franz Mesmer's claims that he could cure illness through "animal magnetism," a committee headed by Benjamin Franklin used a series of placebo-controlled blind trials of test subjects to expose the treatment as a sham.<sup>1</sup>

Since then, experimental design has been largely formalized: researchers randomly assign test subjects to different groups, including a control group, and use statistics to evaluate the outcomes of multiple trials. The earliest double-blind randomized control trial (RCT) of which we are aware was the Nuremberg salt test of 1835, which invalidated a popular homeopathic treatment.<sup>2</sup> A century later, biostatistician Ronald Fisher published his groundbreaking work *The Design of Experiments*. In the ensuing decades, researchers refined RCTs to challenge the fraudulent claims of alternative medicine and various pseudosciences such as telepathy and psychics.<sup>1</sup> RCTs like those carried out by the US Veterans Administration in the 1940s paved the way for modern experimentation and led to new medical advances. Also seminal was the work of Austin Bradford-Hill, a British epidemiologist and statistician who demonstrated the connection between smoking and lung cancer in the 1950s.<sup>3</sup>

By the end of the 20th century, RCTs were commonplace. For example, 114,850 RCT-based medical studies were published between 1990 and 2001—approximately 2.2 percent of all such studies and an 11.2 percent increase per year since the early 1960s.<sup>4</sup> In 1996, efforts to develop unified recommendations for RCTs in medicine culminated in the Consolidated Standards of Reporting Trials (CONSORT) Statement,<sup>5</sup> which was subsequently revised in 2001 and

2010 ([www.consort-statement.org/consort-2010](http://www.consort-statement.org/consort-2010)). In the education field, the US Department of Education's Institute of Education Sciences created the What Works Clearinghouse ([ies.ed.gov/ncee/wwc](http://ies.ed.gov/ncee/wwc)) in 2002 to evaluate "different programs, products, practices, and policies ... to provide educators with the information they need to make evidence-based decisions." As of 2014, 652 of the 11,771 studies the organization had reviewed—about 5.5 percent—used RCTs that met the highest standard of "without reservations."<sup>6</sup>

In short, while not yet a high proportion of the overall literature, RCTs provide an important check and balance on research in a variety of scientific disciplines. Unfortunately, that isn't the case in software engineering.

### References

1. T.J. Kaptchuk, "Intentional Ignorance: A History of Blind Assessment and Placebo Controls in Medicine," *Bull. of the History of Medicine*, vol. 72, no. 3, 1998, pp. 389–433.
2. M. Stolberg, "Inventing the Randomized Double-Blind Trial: The Nuremberg Salt Test of 1835," *J. Royal Soc. of Medicine*, vol. 99, no. 12, 2006, pp. 642–643.
3. H. Marks, *The Progress of Experiment: Science and Therapeutic Reform in the United States, 1900–1990*, Cambridge Univ. Press, 1997.
4. M. Tsay and Y. Yang, "Bibliometric Analysis of the Literature of Randomized Controlled Trials," *J. Medical Library Assoc.*, vol. 93, no. 4, 2005, pp. 450–458.
5. T.R. Elliott, "Registering Randomized Clinical Trials and the Case for CONSORT," *Experimental and Clinical Psychopharmacology*, vol. 15, no. 6, 2007, pp. 511–518.
6. *What Works Clearinghouse: Procedures and Standards Handbook*, version 3.0, Inst. of Education Sciences, US Dept. of Education, 2014; [ies.ed.gov/ncee/wwc/Docs/referenceresources/wwc\\_procedures\\_v3\\_0\\_standards\\_handbook.pdf](http://ies.ed.gov/ncee/wwc/Docs/referenceresources/wwc_procedures_v3_0_standards_handbook.pdf).

the "Historical Evidence Standards in Science" sidebar points out, such standards are commonplace in many fields. However, research suggests that isn't the case in software engineering, especially with respect to programming-language design.

## LACK OF EVIDENCE IN SOFTWARE ENGINEERING

Scholars such as Walter Tichy<sup>3</sup> have criticized the lack of evidence in software engineering for decades, arguing that computer scientists conduct fewer experiments than scientists in

other fields. A recent study by Andrew Ko, Thomas Latoza, and Margaret Burnett<sup>4</sup> supports this assertion. The researchers reviewed papers on software engineering tools over a 10-year period and found that, while 77 percent of the papers included some form of

empirical evaluation, only 27 percent involved human use of a tool. Further, in 62 percent of these studies, the tool users were the authors themselves—an obvious conflict of interest and source of bias. Over the entire decade, just 44 studies had both independent test participants and a control group.

One might assume that work on programming-language design would have higher evidentiary standards given that much of it is funded by the NSF, which explicitly requires analysis of the broader impact of research. However, a detailed survey by Antti-Juhani Kaijanaho<sup>5</sup> of the literature up to 2012 found only 22 randomized controlled trials (RCTs) with human subjects.

Some language-design subfields lack any empirical foundation. For example, Phillip Uesbeck and his colleagues<sup>6</sup> tracked the evidence standards of research presented at the International Conference on Functional Programming from its founding in 1996 to 2014 and found that not one study followed methodological guidelines like those recommended by the What Works Clearinghouse (WWC; [ies.ed.gov/ncee/wwc](http://ies.ed.gov/ncee/wwc)) or the Consolidated Standards of Reporting Trials (CONSORT) Group ([www.consort-statement.org](http://www.consort-statement.org))—for example, having a control group, randomly assigning participants to avoid bias, and using statistics to check assumptions. Nevertheless, this research community has vehemently and successfully argued for the inclusion of functional features in popular programming languages such as Java, C++, and Snap!

## EXISTING RESEARCH ON PROGRAMMING LANGUAGES

Although work on programming languages is limited, researchers have conducted some experiments.

Evidence from one videotaped RCT of programming sessions suggests that static typing is generally more beneficial than dynamic typing, perhaps because with the latter there are no type annotations on bind points and thus it

takes time to discover what to pass to a function.<sup>7</sup>

Empirical research indicates that inexperienced programmers struggle with notations.<sup>8</sup> Another RCT shows that, in C++, novices have particular trouble with lambda expressions.<sup>6</sup>

A large-scale survey of Java compiler errors from students across the world<sup>9</sup> reinforces RCT findings on common typing and syntax mistakes, suggesting a coherence between field and lab studies.

Another investigation found that blocks- and visualization-based programming can be slightly more beneficial than a purely text-based approach in certain regions of code, but not in others.<sup>10</sup>

Beyond these few studies, the vast majority of language features haven't been evaluated using reasonable evidence standards.

## ALTERNATIVES TO EVIDENCE

Without evidence of the broader impact of programming-language changes, on what grounds is the software engineering community arguing for or against particular design choices?

Each year, hundreds of papers are published at conferences such as including PLDI (Programming Language Design and Implementation), OOPSLA (Object-Oriented Programming, Systems, Languages, and Applications), ICFP (the International Conference on Functional Programming), and ECOOP (the European Conference on Object-Oriented Programming). Mathematical approaches play an essential role in many of these papers—for example, the authors describe a new language construct, define its semantics and a type system for it, and then show a proof of type soundness. While useful, mathematic reasoning alone won't reveal whether a particular proposed feature is easy to understand or would actually be useful in practice.

Further, the call for papers at these venues is very odd compared to other scientific gatherings. For example, the calls for papers at the flagship OOPSLA

conference in 2014–2016 stated that acceptable examples of evidence include “proofs, implemented systems, experimental results, statistical analyses, case studies, and anecdotes.” ICFP has a similar statement. No other scientific field known to us accepts anecdotes as a standard of evidence.

In addition, the methodologies used by programming-language scholars are highly suspicious. For example, one common approach that we think lacks merit is the so-called *cognitive dimensions of notations* framework, a set of design principles conceived by Thomas R.G. Green in 1989 and expanded in a 1996 article.<sup>11</sup> According to Google Scholar this influential article has been cited some 500 times, but Green's theory wasn't based on sound empirical evidence—by 1989 there had only been seven programming-language design studies.<sup>5</sup>

Another popular approach called *grounded theory*, which originated in the social sciences in 1967, describes a process to come up with new theories by making observations and reasoning about context. However, in a study of 98 papers at software engineering conferences that applied grounded-theory techniques, only eight actually came up with explicit theories.<sup>12</sup>

In sum, programming-language studies rely on such weak methodological procedures that it's doubtful whether they could even detect fraud, let alone be replicated by other scholars.


## CHECKS AND BALANCES

To address the current methodological regularities in programming-language research, we offer two main suggestions.

First, we implore software engineering conferences and journals to adopt CONSORT as initial standards of evidence for empirical investigations. Such standards would need to be adapted from medicine to software engineering, but they would provide guidance for all stakeholders, including researchers, students, programmers,

and reviewers and editors seeking higher-quality evidence. Mathematical reasoning will always remain central to programming-language science, but we need significantly more input from developers and users.

Second, we should adopt similar strategies to those of the WWC in regard to peer review. In our own experience, most reviewers have a sound grasp of theoretical principles but lack the expertise to evaluate empirical work. WWC uses reviewer certification, which might or might not be appropriate for software engineering, but we must do something to stem the publication of methodologically tenuous work.

**S**ubstantial industry and government investments in software are at risk from changes in the underlying programming languages, despite the fact that such changes have no empirically verified benefits. One way to address this long-standing problem is to establish rigorous evidence standards like those in other sciences, beginning with the adaptation of well-established methodological guidelines from medicine and education to software engineering. 

## ACKNOWLEDGMENTS

This work was partially funded by the NSF under grant CNS-1440878.

## REFERENCES

1. "Gartner Says Worldwide Software Market Grew 4.8 Percent in 2013," press release, Gartner, 31 Mar. 2014; [www.gartner.com/newsroom/id/2696317](http://www.gartner.com/newsroom/id/2696317).
2. M. Smith, "Computer Science for All," blog, 30 Jan. 2016; [obamawhitehouse.archives.gov/blog/2016/01/30/computer-science-all](http://obamawhitehouse.archives.gov/blog/2016/01/30/computer-science-all).
3. W.F. Tichy, "Should Computer Scientists Experiment More?," *Computer*, vol. 31, no. 5, 1998, pp. 32–40.
4. A.J. Ko, T.D. Latoza, and M.M. Burnett, "A Practical Guide to Controlled Experiments of Software Engineering Tools with Human Participants,"

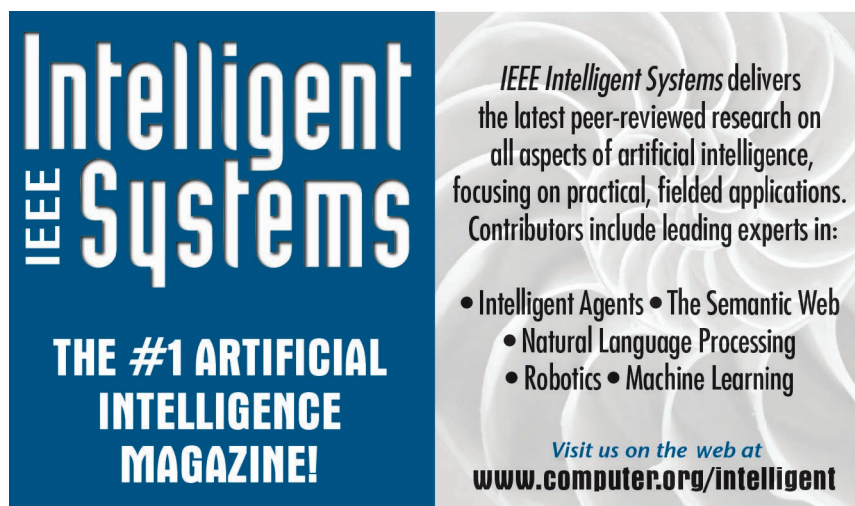
*Empirical Software Eng.*, vol. 20, no. 1, 2015, pp. 110–141.

5. A.-J. Kaijanaho, "Evidence-Based Programming Language Design: A Philosophical and Methodological Exploration," PhD dissertation, Univ. of Jyväskylä, 2015.
6. P.M. Uesbeck et al., "An Empirical Study on the Impact of C++ Lambdas and Programmer Experience," *Proc. 38th Int'l Conf. Software Eng. (ICSE 16)*, 2016, pp. 760–771.
7. S. Endrikat et al., "How Do API Documentation and Static Typing Affect API Usability?," *Proc. 36th Int'l Conf. Software Eng. (ICSE 14)*, 2014, pp. 632–642.
8. A. Stefik and S. Siebert, "An Empirical Investigation into Programming Language Syntax," *ACM Trans. Computing Education*, vol. 13, no. 4, 2013, article no. 19.
9. A. Altmirri and N.C.C. Brown, "37 Million Compilations: Investigating Novice Programming Mistakes in Large-Scale Student Data," *Proc. 46th ACM Technical Symp. Computer Science Education (SIGCSE 15)*, 2015, pp. 522–527.
10. D. Weintrop and U. Wilensky, "Using Commutative Assessments to Compare Conceptual Understanding in Blocks-Based and Text-Based Programs," *Proc. 11th Ann. Int'l Conf. Int'l Computing Education Research (ICER 15)*, 2015, pp. 101–110.
11. T.R.G. Green and M. Petre, "Usability Analysis of Visual Programming Environments: A Cognitive Dimensions Framework," *J. Visual Languages and Computing*, vol. 7, no. 2, 1996, pp. 131–174.
12. K.-J. Stol, P. Ralph, and B. Fitzgerald, "Grounded Theory in Software Engineering Research: A Critical Review and Guidelines," *Proc. 38th Int'l Conf. Software Eng. (ICSE 16)*, 2016, pp. 120–131.

**ANDREAS STEFIK** is an associate professor of computer science at the University of Nevada, Las Vegas. Contact him at [stefika@gmail.com](mailto:stefika@gmail.com).

**STEFAN HANENBERG** is a scientific assistant at Paluno—The Ruhr Institute for Software Technology, University of Duisburg-Essen. Contact him at [stefan.hanenberg@uni-due.de](mailto:stefan.hanenberg@uni-due.de).

*This article originally appeared in Computer, vol. 50, no. 8, 2017.*



**IEEE Intelligent Systems**

**THE #1 ARTIFICIAL INTELLIGENCE MAGAZINE!**

*IEEE Intelligent Systems delivers the latest peer-reviewed research on all aspects of artificial intelligence, focusing on practical, fielded applications. Contributors include leading experts in:*

- Intelligent Agents • The Semantic Web
- Natural Language Processing
- Robotics • Machine Learning

*Visit us on the web at*  
[www.computer.org/intelligent](http://www.computer.org/intelligent)