Usability of Programming Languages

Special Interest Group (SIG) meeting at CHI'2016

(see also: **www.programminglanguageusability.org**)

Notes taken at the SIG Meeting

Table of Contents

A. **Articles**, papers, blogs, and other references that Report on the Usability of Programming Languages (with full citations and hyperlinks)

Entries in the following sections can refer to these references (just "Insert" a "Bookmark" into the start of the reference, and you can cross reference it below using "Insert" "Link…" "Bookmark").

1. Brad A. Myers, Andrew J. Ko, Thomas D. LaToza, and YoungSeok Yoon. "Developers are Users Too: Human Centered Methods to Improve Software Development," *IEEE Computer*, Special issue on UI Design, accepted for publication in vol.49, issue 7, July, 2016. preprint pdf
2. John F. Pane, Chotirat "Ann" Ratanamahatana, and Brad A. Myers, "Studying the Language and Structure in Non-Programmers' Solutions to Programming Problems",*International Journal of Human-Computer Studies (IJHCS)*. Special Issue on Empirical Studies of Programmers, vol. 54, no. 2, February 2001, pp. 237-264. http://www.cs.cmu.edu/~pane/IJHCS.html
3. J.F. Pane, B.A. Myers, and L.B. Miller, "Using HCI Techniques to Design a More Usable Programming System," *2002 IEEE Symposia on Human Centric Computing Languages and Environments (HCC'2002)*. Arlington, VA, September 3-6, 2002. pp. 198-206. http://www.cs.cmu.edu/~pane/handsdesign.html

4. John Pane and Brad Myers, "Tabular and Textual Methods for Selecting Objects from a Group," *IEEE Symposium on Visual Languages*, VL'2000, Seattle, Washington, September 10-14, 2000. pp. 157-164. html and local pdf
5. John Pane and Brad Myers. *Usability Issues in the Design of Novice Programming Systems,* Carnegie Mellon University School of Computer Science Technical Report CMU-CS-96-132. and Human Computer Interaction Institute Technical Report CMU-HCII-96-101, August, 1996. http://www.cs.cmu.edu/~pane/cmu-cs-96-132.html http://reports-archive.adm.cs.cmu.edu/anon/1996/CMU-CS-96-132.ps
6. Mary Beth Kery, Claire Le Goues, Brad A. Myers, "Examining Programmer Practices for Locally Handling Exceptions", *Mining Software Repositories* (MSR'2016) Mining Challenge Track, Austin, TX, USA, 14-15 May, 2016. To appear.
7. Christopher Bogart, Margaret Burnett, Scott Douglass, Rachel White, Hannah Adams, "Designing a debugging interaction language: An initial case study in Natural Programming Plus", *ACM Conference on Human Factors in Computing Systems (CHI)*, May 2012, pp. 2469-2478. local pdf
8. Christopher Bogart, Margaret Burnett, Allen Cypher, and Christopher Scaffidi, End-User Programming in the Wild: A Field Study of CoScripter Scripts, IEEE Symposium on Visual Languages and Human-Centric Computing, Herrsching am Ammersee, Germany, Sept. 2008, pp. 39-46. local pdf
9. E. M. Wilcox, J. W. Atwood, M. M. Burnett, J. J. Cadiz, C. R. Cook, Does Continuous Visual Feedback Aid Debugging in Direct-Manipulation Programming Systems?, ACM Proceedings CHI'97: Human Factors in Computing Sysntems, Atlanta, GA, 258-265, Mar. 22-27, 1997. html
10. Sherry Yang, Margaret Burnett, Elyon DeKoven, and Moshe Zloof, Representation Design Benchmarks: A Design-Time Aid for VPL Navigable Static Representations, Journal of Visual Languages and Computing, Oct/Dec 1997, 563-599. local pdf
11. Jason Dagit, Joseph Lawrance, Christoph Neumann, Margaret Burnett, Ronald Metoyer, and Sam Adams, Using Cognitive Dimensions: Advice from the Trenches, Journal of Visual Languages and Computing 17(4), 302-327, August 2006. local pdf
12. Okon, Hanenberg, Can We Enforce a Benefit for Dynamically Typed Languages in Comparison to Statically Typed Ones? A Controlled Experiment, ICPC 2016
13. Fischer, Hanenberg, An Empirical Investigation of the Effects of Type Systems and Code Completion on API Usability using TypeScript and JavaScript in MS Visual Studio, DLS 2015 [link]
14. Endrikat, Hanenberg, Robbes, Stefik, How do API documentation and static typing affect API usability?, ICSE 2014 [link]
15. Petersen, Hanenberg, Robbes, An empirical comparison of static and dynamic type systems on API usage in the presence of an IDE: Java vs. groovy with eclipse, ICPC 2014 [link]
16. Spiza, Hanenberg, Type names without static type checking already improve the usability of APIs (as long as the type names are correct): an empirical study, AOSD 2014 [link]

17. Hanenberg, Kleinschmager, S.Robbes, R.Tanter, Stefik: An empirical study on the impact of static typing on software maintainability, ESE 2014 [link]
18. Hoppe, Hanenberg: Do developers benefit from generic types? An empirical comparison of generic and raw types in Java, OOPSLA 2013 [link]
19. Kleinschmager, Hanenberg, Robbes, Tanter, Stefik: Do static type systems improve the maintainability of software systems? An empirical study. ICPC 2012 [link]
20. Mayer, Hanenberg, Robbes, Tanter, Stefik: An empirical study of the influence of static type systems on the usability of undocumented software. OOPSLA 2012 [link]
21. Hanenberg, S., “A chronological experience report from an initial experiment series on static type systems,” ESCOT 2011 [link]
22. Stuchlik, Hanenberg: Static vs. dynamic type systems: An empirical study about the relationship between type casts and development time. DLS 2011 [link]
23. Hanenberg: Doubts about the Positive Impact of Static Type Systems on Programming Tasks in Single Developer Projects - An Empirical Study. ECOOP 2010 [link]
24. Hanenberg: An experiment about static and dynamic type systems: doubts about the positive impact of static type systems on development time. OOPSLA 2010 [link]
25. Hanenberg,What is the Impact of Static Type Systems on Programming Time? Preliminary Empirical Results. PLATEAU 2009 [link]
26. Dan Luu. Static vs. dynamic languages: a literature review. Blog post. http://danluu.com/empirical-pl/ [Discussed on Lambda the Ultimate in November 2015. http://lambda-the-ultimate.org/node/5286]
27. Weintrop, D. & Wilensky, U. (2015). Using Commutative Assessments to Compare Conceptual Understanding in Blocks-based and Text-based Programs, ICER’15 [link]
28. Weintrop, D. & Wilensky, U. (2015). To Block or not to Block, That is the Question: Students’ Perceptions of Blocks-based Programming. In Proceedings of the 14th International Conference on Interaction Design and Children. New York, NY, USA: ACM. [link]
29. Weintrop, D. & Wilensky, U. (2017). Comparing Blocks-based and Text-based Programming in High School Computer Science Classrooms. *Transactions on Computing Education (TOCE)*, 18(1), 1-25. [link]
30. Michael Coblenz, Joshua Sunshine, Jonathan Aldrich, Brad Myers, Sam Weber, and Forrest Shull. "Exploring Language Support for Immutability." The 38th International Conference on Software Engineering (ICSE 2016), Austin, TX, May 14 - 22, 2016.
31. Michael Coblenz, Jonathan Aldrich, Brad Myers and Josh Sunshine. "Considering Productivity Effects of Explicit Type Declarations", The Fifth Workshop on Evaluation and Usability of Programming Languages and Tools (PLATEAU 2014), at SPLASH 2014, 21 Oct 2014, Portland, OR
32. Dimitar Asenov and Otmar Hilliges and Peter Müller: “The Effect of Richer Visualizations on Code Comprehension” CHI `16 [link]
33. Leonel Morales Diaz: “Programming Languages as User Interfaces” MexIHC’10 Proceedings of the 3rd Mexican Workshop on Human Computer Interaction, 2010 [link]
34. Paul Graham, “Hackers and Painters”, essay, May 2003, [link]

B. **What is Known** about programming language usability as a result of those studies

1. Mental models: in the designer's mind there is a mental model of the computer or computational system as a programmable artifact, that model is reflected in the structures and words used in the language. Programmers form a mental model based on the structures and words used in the language. Both mental models may or may not match. Matching of mental models seems to be a desirable feature of programming languages, is it?
2. Programming languages are designed around a programming paradigm, there are several, none of which can be proclaimed the "most usable" paradigm. On the other hand, usability criteria may be "paradigm-dependant" so usability evaluation could make more sense if programming paradigm is considered.
3. As with the transition from "character-based" user interfaces (CUIs) to GUIs, there is nowadays an ongoing transition in programming languages from keyboard-typed languages to block programming (drag and drop). Arguably GUIs never achieved universal adoption, several CUIs still exist and enjoy popularity in certain contexts. Block programming will become popular and widely adopted, but as with GUIs, it will never reach universal adoption. Lessons from CUI-to-GUI transition apply: just because an UI is graphical it doesn't mean it is usable, GUIs are not inherently usable, in fact GUIs can be quite unfriendly.

C. **Methods** that can help with Programming Language Usability

For each, list name of method, what it can be used for, references that discuss how it can be used.

1. Contextual Inquiry field studies
    - To better understand programmers' real problems
    - Differences between intended use of structures and commands and actual use. Unexpected, innovative uses can be identified.
2. Natural Programming Elicitation
    - To better understand how the target developers think about the tasks
3. Natural Programming Plus: for language designers who are not HCI experts
4. A/B Testing
    - Comparison of usability of different languages in regard to the codification of a particular algorithm or solution for a problem

5. Randomized Controlled Trials
    - Performance of programmers solving a problem or coding an algorithm under different conditions (different programming languages), performance measured as time to complete, errors and corrections count, number of variables used
6. Cognitive Dimensions of Notations
7. Other Analytical/Predictive methods e.g. Cognitive Walkthrough
8. Representation Design Benchmarks
9. Commutative Assessments (comparing blocks and text syntax)
10. Instrumented tooling/Log analysis
11. Mining software repositories: gather evidence in the context of large-scale software development
12. Ethnographic studies of End User Programmers
13. Think aloud studies
14. Interviews and surveys
15. Learn from existing GUI application design
    - Many apps are domain-specific
    - What corresponds to Photoshop vs Illustrator vs …?
16. Experimental design templates
17. Power Law of Practise
18. Replication packet (or artifact evaluation as used in PL/SE conferences)

D. What **Needs** to be Studied?

What in particular needs to be studied? What are parts of programming languages that are particularly difficult with respect to usability? For each, add sub-bullets about why it is a problem, and any references substantiating this issue.

1. Error handling
    - Still using the same designs from the beginning of programming - error value returns and exceptions, which have been shown to be difficult for novices and experts
    - Paper [Kery 2016] above shows that novices and experts do exception handlers poorly
2. What can we learn from Cognitive Psychology
    - Are there innate human cognitive abilities, and can they be used to help direct programming language design
    - Novice vs Expert
    - Anderson - Cognitive Psychology and its Implications
3. Order of teaching programming
    - Look at Matthias Felleisen “How to design programs” and his talk on what he thinks should be in book 2.

4. Language features instead of whole languages (but, eventually also interaction of language features)

E. Community challenges to address:

a. We seem to have managed to assemble perhaps the only largely male-dominated room at CHI. How can we make PL research more equitable and diverse?
    i. Take lessons from universities that are succeeding in attracting female students into their CS programs, their strategies are applicable here.
b. What causes people to drop out of learning programming, and what can be done about theses. (variables, loops, higher order functions, recursion)
    i. The above question assumes that these features are the cause; why?
    ii. First year CS should have numbers on these. These are just what I anecdotally have seen. (By the way I'm coming from a spreadsheets as programming background).
c. What should a newcomer read in order to get started on this work and/or join the community?

F. People Interested:

Please enter your name, affiliation and email, in case we create a Google Group or email list. <Update from 2018: Nothing has happened since 2016, so any such list is unlikely, but you might be interested in who is listed as interested in this topic.>

NAME	AFFILIATION	EMAIL ADDRESS
Brad A. Myers	Carnegie Mellon University	bam@cs.cmu.edu
Dimitar Asenov	ETH Zurich	dimitar.asenov@inf.ethz.ch
Gary Miller	University of Technology Sydney	miller.garym@gmail.com
Margaret Burnett	Oregon State University	burnett@eecs.oregonstate.edu
Franklyn Turbak	Wellesley College	fturbak@wellesley.edu
David Weintrop	University of Maryland	weintrop@umd.edu
Thomas Prokosch	University of Innsbruck, Austria	thomas-plu@nadev.net
Luke Church	University of Cambridge/Google	luke@church.name
Michael Coblenz	Carnegie Mellon University	mcoblenz@cs.cmu.edu

Andrew Head	UC Berkeley	Andrewhead@berkeley.edu
Lea Verou	MIT	leaverou@mit.edu
Jun Kato	AIST, Japan	i@junkato.jp
Poorna Talkad Sukumar	University of Notre Dame	ptalkads@nd.edu
Matt Kulukundis	Google - C++ Libraries Team	kfm@google.com
Matthias Hauswirth	Università della Svizzera italiana	Matthias.Hauswirth@gmail.com
Michael Rohs	University of Hannover	michael.rohs@hci.uni-hannover.de
Molly Feldman	Cornell University	molly@cs.cornell.edu
Antti-Juhani Kaijanaho	University of Jyvaskyla	antti-juhani.kaijanaho@jyu.fi
Sandeep K Kuttal	University of Tulsa	sandeep-kuttal@utulsa.edu
Michelle Ichinco	Washington University in St. Louis	michelle.ichinco@wustl.edu
Leonel Morales	Universidad Francisco Marroquin	litomd@ufm.edu
Vinson Chuong	Pivotal Labs	vinsonchuong@gmail.com
Andrew Macvean	Google	amacvean@google.com
Parmit Chilana	University of Waterloo	pchilana@uwaterloo.ca
Amy J. Ko	University of Washington	ajko@uw.edu
Rohit Ramesh	University of Michigan	rohitram@umich.edu
YoungSeok Yoon	Google	youngseokyoon@google.com
Sridhar Chimalakonda	International Institute of Information Technology - Hyderabad	sridhar_ch@research.iiit.ac.in
Yasaman Sefidgar	University of Washington	einsian@cs.washington.edu
Meadhbh Hamrick	Amazon	OhMeadhbh@gmail.com
Harikrishnan G.	Atlas Copco AB.	hkrish.etr@gmail.com
Gudmund Grov	Heriot-Watt University	G.Grov@hw.ac.uk
Ramrao Wagh	Goa University	ramrao@unigoa.ac.in

Gary Lupyan	University of Wisconsin-Madison	glupyan@gmail.com
James Evans	University of Chicago	jevans@uchicago.edu
Claire Kearney-Volpe	New York University	claire.kv@nyu.edu
Omar Shehab	IonQ, Inc.	shehab@ionq.co
Jürgen Cito	MIT	jcito@mit.edu
Alexander Zeier	University of Applied Sciences Darmstadt	alexander.zeier@h-da.de

General Notes:

What is known discussion:

Lyn Turbak:

1. Lots of claims about blocks being easier to use, but not a lot of evidence.
2. Comparisons between text and visual programming could help out sort out the truth

Margaret Burnett:

1. Big fan of diversity, in the context that people are different from one another
2. The gentle slope idea might be helpful
3. In coScripter, there is the "you" construct. You can put it in anywhere, which basically allows the language to pause. For example, you might use it to "wait" for a web page to load, which could help people at different ability levels.

Brad:

1. Logical operators are not well understood in programming language
2. The pane study was with children, which may have made an impact

Philip Wadler:

1. Studying functional programming is hard, with studies not being particularly clear on what the answer is
2. Asking what people likes isn't necessarily a good way to evaluate languages

Amy Ko:

1. We should also be looking at how people are taught programing languages

Patrice:

1. Interpreted vs. non-interpreted is an interesting question

Unknown:

1. We might be arguing about the wrong thing. Cognitive dimensions is helpful
2. Others disagree that cognitive dimensions can provide us insight

Methods:

1. One reason why ECMA 6 didn't use evidence is because it takes too long. Same with C++ 14.
2. We need robust methods that are fast.
3. With expert evaluations, who are the experts? Brad says the methods are designed for HCI experts to use.
4. One method I think is missing from this list is corpus based methods. With scaling issues, or with undergraduates, we have to ask how we are going to scale beyond this.
5. Randomized controlled trials are very expensive for this kind of thing - we can use with other types of evidence such as instrumented tooling to get externally valid evidence.
6. One person is concerned that we haven't used the word taste or aesthetic, but others seem unconcerned
7. One thing that majorly impacts usability is tools. The semantics of the language impacts what kind of tooling is possible.
8. Dykstra considered goto harmful, should we try to prove that?
9. Brad says it is difficult to make claims that are truly universal.

What needs to be studied:

1. We need to think about semantics
2. We need to think more about the metrics that could be used

What to do next:

1. Amy Ko: We should do a Dagstuhl
2. We need participation from people in industry
3. We need to think about what the right "order" is for teaching programming language constructs
4. We need a book on what tradeoffs exist and why they are important.
5. Should we look outside of our community to help --- perhaps the computer science education community, but also perhaps psychologists

6. This is a large community with immature tools. A nice outcome might be a larger collaborative project on the topic.

Mat Kulukundis:

1. C++ Reading team with about 150 engineers. They might be able to take surveys and such.
2. Might be able to collaborate with some of the folks working on Go/Google
3. Fibers is a Go-like C++ interface for concurrency

Carl:

Inconsistency Robustness - Name of the book

Give or take straw poll (Rough, Stefik could not immediately get an accurate count):
75% academia
25% industry