Human Aspects of Programming: Understanding Programming Languages and Tools More Effectively

Craig Anslow
craig@ecs.vuw.ac.nz
craiganslow
http://homepages.ecs.vuw.ac.nz/~craig/
The Mythical Man-Man-Month

The beginning of the awareness of the human aspects of software engineering.

In the preface to the 20th Anniversary Edition, Brooks writes that he is surprised that The Mythical Man-Man-Month is popular even after 20 years. Now 40+ years!

The Mythical Man Month - Fred Brooks (1975, revised 1995)
Human Aspects of Software Engineering

Software is created for and with a wide range of stakeholders, from customers to management, from value-added providers to customer service personnel. These stakeholders work with teams of software engineers to develop and evolve software systems that support their activities. All of these people and their interactions are central to software development. Thus, it is crucial to study the dynamic and frequently changing Human Aspects of Software Engineering.

CHASE Workshop @ ICSE
http://www.chaseresearch.org/

IEEE VL/HCC Conference
https://sites.google.com/site/vlhcc2017/

Tomayko, Hazzan 2004
British Airways says IT chaos was caused by human error

The boss of British Airways' parent company says that human error caused an IT meltdown that led to travel chaos for 75,000 passengers.

Human aspects of programming concerning *usability & effectiveness*: programming languages, tools for programming and understanding, and development processes.
HAOP - Research Areas

1. Evaluation and Usability of Programming Languages and Tools

2. Empirical Studies of Programming Languages and Programmers

3. Design of Programming Languages Community Perspective
• Programming languages exist to enable programmers to develop software effectively

• How efficiently programmers can write software depends on the usability of the languages and tools that they develop with

• **Purpose:** Discuss methods, metrics and techniques for evaluating the usability of PL and language tools.

• **Benefits:**
  - make programs easier to read, write, and maintain
  - allow programmers to write more flexible and powerful programs
  - restrict programs to make them more safe and secure
PLATEAU Reflection

• Tools and Environments
• Languages
• Program Comprehension
• Libraries and APIs
• DSL
• Metrics
• Empirical Studies: Surveys and Questionnaires
Summary: Programming Tools

- How do we evaluate tools more effectively?
- Often help with the design process
- Typically formal studies performed at end of prototype
- Students are really only ok for novice / introductory programming concepts
- Hard to get professional programmers
- Often in controlled labs so limited time
- Hard to have a controlled variable
- Need more longitudinal studies in the wild
- Maybe community standards for evaluation
HAOP - Research Areas

1. Evaluation and Usability of Programming Languages and Tools

2. Empirical Studies of Programming Languages and Programmers

3. Design of Programming Languages Community Perspective
Qualitas Corpus

- Since 2005
- Curated collection of software systems
- Used for empirical studies of code artefacts
- **Goal**: provide a resource that supports reproducible studies of software
- 112 systems, 15 systems with 10 or more versions, and 754 versions total
- A large-scale empirical study of practitioners' use of object-oriented concepts (ICSE 2010) - **3785 responses!**
National Java Resource

• Our goal is to create a Normalized Java Resource (NJR) that will speed up innovation in the area of software tools. Those tools include security enhancers, bug finders, and code synthesizers, all of which can benefit greatly from access to Big Code. The vision is a diverse collection of 100,000 normalized Java projects that is executable, scriptable, and searchable. The project will enable faster building and evaluation of a variety of tools. A tool developer can rely on that many popular tools already work for our 100,000 Java projects, and then build their own tools on top.

• Jens Palsberg (UCLA) and Cristina Lopes (UC Irvine) PIs
Understand why programmers make the choices they do, especially when practice contradicts research
Programming Languages Choice: Research Problem

• Not much research has been done into why programmers do what they do, L. Meyerovich 2013
• Previous studies show development using static languages results in fewer errors and faster development speed, L. Paulson, 2007
• More programmers turning to dynamic languages, Pypl.github.io, 2017
• Investigate why programmers opt to use dynamic or static languages and how they make these choices
Grounded Theory

- Gather participants through industry and university contacts
- Interview participants
- Transcribing the interviews
- Analysing the data for codes
- Aggregating these codes into concepts and categories
- Form the theory from these concepts and categories
Findings: Experience

**Static**

**Speed**
- Static build/compile times slow down large projects
- Dynamic languages good for quick starts

**Errors**
- Compilers provide better error checking
- No types means lower learning curve

**Structure**
- Static languages enforce more structure
- Dynamic Languages provide more flexibility

© Craig Anslow 2018
Findings: *Attitudes*

- Static Languages partisanship
- Developers with more experience tend to prefer static languages
- Switching between static and dynamic languages is not an issue
Findings: *Choice*

- Project languages often pre-selected
- Languages chosen based on familiarity
- Tooling, IDE, and library support heavily influence programming language choice
Findings: Relationships

- Choice
- Attitudes
- Experience

- Provides
- Informs
- Shapes
Summary: Empirical Studies of Programming Languages

- Need more studies done of programming languages exploring features
- Need more studies on multiple programming languages
- Need to conduct more interviews and studies on how programmers use and adopt programming languages
HAOP - Research Areas

1. Evaluation and Usability of Programming Languages and Tools

2. Empirical Studies of Programming Languages and Programmers

3. Design of Programming Languages Community Perspective
How many of you are programming language designers?
Short survey of programming language designers (n=20).

Caveat (My Research Group): Grace, Whiley, 42, DeepJava, Wyvern

**RQ1.** How do you decide what to put you in your programming language (i.e. what is the design process)?

**RQ2.** Have you ever conducted formal human experiments with programmers of your programming language? If so what was the outcomes?

**RQ3.** Do you consider doing human experiments with your programming language to get a better design an effective technique?
RQ1) PL Design Process

• “Intuition.” Phil Wadler (Haskell, XQuery, Orwell, JavaGenerics, …)

• “A combination of (a) history (b) semantics (c) user appeal (d) pedagogic needs and (e) observational studies.” Shriram Krishnamurthi (Racket, Pyret, …)

• “You create a mental model of the target applications and their issues (e.g. web programming), form a hypothesis about what is important in that space and design a language. Basically aim and throw a dart!” Jan Vitek (Thorn, FastR, …)
Responsibility for adding a feature is with the language team (see https://www.rust-lang.org/en-US/team.html, 7 people). The process is usually an informal pre-RFC which might be posted on our developer forums (https://internals.rust-lang.org/) or might be discussed in a GitHub issue or rarely discussed in a meeting, discussion here is with stakeholders, rather than the language team and might not involve anyone from the language team, however, in **practice almost any successful idea has a champion on the language team even at this stage**. The next stage is a formal RFC, see https://github.com/rust-lang/rfcs for example RFCs (which are pull requests until accepted) and an outline of the RFC process. RFCs are discussed by the whole Rust community, i.e., they are open for anyone to comment. Typically an RFC goes through a lot of modifications at this stage. Once ready for a decision we enter a 'final comment period' where the language team indicate whether they think an RFC should be accepted or closed, that discussion can be changed during the final comment period. Decisions must be unanimous.

Once accepted an RFC is implemented in the compiler. It starts as an unstable feature which can only be used with the nightly (cf stable) compiler and which authors must explicitly opt-in to. Whilst unstable, features are often tweaked. After some time (between 8 weeks and multiple years depending on urgency, complexity, etc) the feature will be nominated for stabilisation, at this point we require another unanimous vote from the language team and another 'final comment period' for the community to register any objections. If all is well after the fcp (which is either 10 or 14 days, I forget exactly), the feature becomes stable and a full part of the language - it then takes up to 12 weeks for the feature to 'ride the trains' from nightly to beta to stable.

*Nick Cameron (Rust)*
RQ1) PL Design Process

“Complex question. I can write pages on this. At some point I meant to, but then there are always other more pressing projects to work on. Basically, I started with a very simple programming model, later we came to call it the APGAS model. This model has a notion of places — a program runs over many places, a place abstracts away from memory and collection of activities operating on that memory — a notion of fine grained concurrency (activities created by async and dynamically scheduled by a user-space scheduler, a la Cilk, and introduced by the async keyword), a dual notion of finish (this was one of my original contributions — this is much more general than Cilk’s construct, it is not tied to recursion in any way), and a notion of at to switch between places. We had non-presumptive exceptions from day one — I had seen their power with Java. We had a lot of detailed discussions and debates about concrete language features and how best to get them in the language. Some very heated debates. ”

Vijay Saraswat (X10)
“I ask: why am I designing this language? Then I include the **minimal set of features to meet my goal**. For example, if my language is designed to support distributed and mobile applications, I look for the minimal set of feature to support distribution and mobility. Any feature added to the language should remove more complexity from a typical program (not from a synthetic program designed to exploit the feature).” *Andrew Black (Emerald, Grace, …)*
“We set out our goals pretty clearly from the beginning. We were interested in designing an object-oriented programming language with target audience novice programmers that would make it easy for students to transition to other OO languages. That mantra gets repeated many times (especially by me) in each of our design meetings.” Kim Bruce (Grace, …)
RQ1) PL Design Process

“I ask the following questions:

Would adding a new feature/mechanism introduce redundancy?

Is there a way in which the same purposes can be served by using existing mechanisms and supporting libraries?

Is the feature/mechanism fundamental, i.e., is it suitable to be one of very few fundamental building blocks?

Is the feature/mechanism intuitive, i.e., will human programmers be able to use it effectively?” Thomas Kuehne (DeepJava)
“Simple trial and error over a long period of time. Whiley has been in development since 2009, and I'm still tweaking the syntax. Partly this is driven by implementation constraints. Partly its driven by my feelings having written some (still small) programs in it.” *David Pearce (Whiley)*
“Trying to prioritise security and programmer productivity above or else. Experience in many other languages and the opinion of the language design meeting attendees on how they would affect the overall goal described in our manifesto of design driven assurance.” Alex Potanin (Wyvern)
RQ2) Formal Human Experiments of Programming Languages

Leprechaun of Programming Languages Evaluation: sample size

STOP 2015 Workshop Panel: “Types for an Untyped World”
RQ2) Formal Human Experiments of Programming Languages

• “Yes. Numerous papers of mine touch on various aspects of the user experiments (error messages, semantics, etc.). But much of the work is not published because the goal was to improve the language, not to publish papers. A great example is the very existence, and then the constant shaping and re-shaping, of language levels in DrRacket.” Shriram

• “No. the only language that I know did is X10” Vitek

• “No.” Wadler

• “No.” Black

• “No.” Kuehne
“Vivek Sarkar ran a user experiment (this has been reported in the literature) in 2005 or so, at the Pittsburgh Supercomputing Center. We later did studies with students at Rice as well. In general we found that people quite liked the basic concepts of X10, compared with MPI and OpenMP.” Vijay Saraswat (X10)
“We have not, though it is something we've talked about doing repeatedly. **We basically lack the experience and bandwidth to do so, plus some concern about getting useful data.**”

*Cameron (Rust)*
RQ2) Formal Human Experiments of Programming Languages

•“No. The closest would be the use of Whiley in SWEN224 where, for a few years, around 200 students used the language. I collected some informal feedback from that.” Pearce

•“Nothing formal, but I had some student doing some exercises, and seems like they can look to examples and use them to cut/paste/adapt solutions without the need to understanding the inner working, exactly as it should happen for all usable languages (that is, if you need understanding of a tool to use such tool, that tool is a failure).” Marco Servetto (42)
RQ3) Are Human Experiments Effective?

“Yes. See the following.”

Usability of Programming Languages
Special Interest Group (SIG) meeting at CHI’2016

04. – 09. February 2018, Dagstuhl Seminar 18061
Evidence About Programmers for Programming Language Design
https://www.dagstuhl.de/de/programm/kalender/semhp/?semnr=18061

Wadler
“Yes I do. John Richards is the right guy to talk to at length about this. He had one person (Jonathan Brezin), new to X10, spend a lot of time learning about X10 and keeping detailed notes about the usability of the language.”

*Saraswat*
RQ3) Are Human Experiments Effective?

“To a limited extent. Our recent paper should be sobering to anyone who is overly in love with this process. Languages are not only human artifacts; they are also formal ones. So the human input needs to be carefully balanced against formal needs as well, and they have to be made to work in harmony. Sometimes the formal techniques can foresee problems that human processes cannot (the above paper offers some disconcerting examples of this sort), so any human-centric process needs to have explicit safeguards against going astray in this way.” Shriram

http://cs.brown.edu/~sk/Publications/Papers/Published/tpk-crowdsoure-lang-design/
“It has great potential, but it’s hard to get right. There are so many possibilities for confounding factors. Take the experimental work that Schärli and I did to validate traits. We concluded that there was no downside to fine-grained trait composition. And that was true given the programming environment we had — which NOONE has since replicated. I don’t think that it’s a reasonable conclusion without the environment.” Black
“Maybe. I think it's something I'm interested to try and see how it goes, at least. Ultimately though, Whiley is not about designing the most usable language ever. It's meant to be a standard language with verification bolted on.”

*Pearce*
“It is not something I know how to do. I am doubtful that we can do it before the fact. and infer conclusions that will apply to a design that isn’t realized yet. All of the studies that I have seen in press are fishy (e.g. small size, only students, problems that are not representative …)” Vitek
“Not for language design. I think, we need to learn way too much basic things before it could be useful. And, it can’t be done in the setting I am working with. I suppose you only really can do this in a useful way in large software companies. So, yeah, other Stefans (Hanenberg) can do that!” Stefan Marr
"I've been out of PL design for quite a long time - I wasn't good enough at it IMHO. That said, I'm not aware of any meaningful impact of HCI/SE research on PL design. The two communities are very far away from each other, and I don't really see that changing in a meaningful way any time soon.” Laurie Tratt
“I've only designed language extensions, never a whole language from scratch. Human experiments are great but it's basically impossible to get real non-student programmers using a language for real for enough time to be truly convincing. I think there is no research that does that.” Patrick Lam
“I'm not exactly sure what you mean by human experiments, nor to get a better design an effective technique. We have never done any formal experiments. We basically rely on good design to ensure a feature is sound, useful, and ergonomic, followed by lots of iteration on the design both before and after implementation where we can get feedback from the entire community.” Cameron (Rust)
RQ3) Are Human Experiments Effective?

• **It would be nice**, but for the type of project I was working on it is hard to see how the effort of a user experiment would be justifiable.” *James Cheney*

• “**Such experiments can be a very good tool to prove/disprove hypotheses regarding the suitability of ideas/concepts.** However, they need to be very well designed, otherwise they are useless or (as they are often) misleading.” *Kuehne*

• “**We want to do them** as Jonathan Aldrich will discuss in the Daghstul you are going to next week.” *Potanin*
• Most (almost all) programming language designers don’t use human centred design methods (qualitative or quantitative). However, they think it is a good idea but hard to do in practice!

• How do we know when we have added usable features into a programming language?

• How do we make PL designers more aware of human centred design methods?

• How do we convince / get more PL designers to adopt human centered design methods?

• How big should our sample size be when evaluating the design of a programming language?

• Conduct a large survey of perspectives on the design of programming languages
Human Aspects of Programming

Need to study human aspects of programming to develop more **usable** and **effective**: programming languages, tools for programming and understanding, and development processes.

Craig Anslow
craig@ecs.vuw.ac.nz
craiganslow
http://homepages.ecs.vuw.ac.nz/~craig/