Need Finding for PL

Reading Reflection

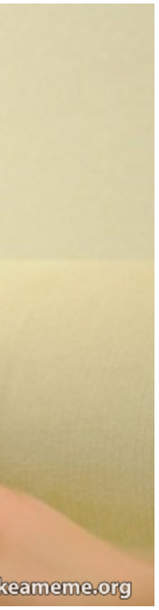
Discuss in groups

- Think back to before you learned about need finding (whether that was from today's readings or in the distant past). Did you instinctively use need finding techniques to find problems to work on? How?
- If/when you used need finding by instinct, did you mostly focus on users with skills like yours?
- When was the last time you talked to someone and came away with an idea for a new library, abstraction, programming tool, or programming environment?
- Can you think of any dangers of using techniques borrowed from ethnography and anthropology?

Finally time to talk about actually conducting user studies!!

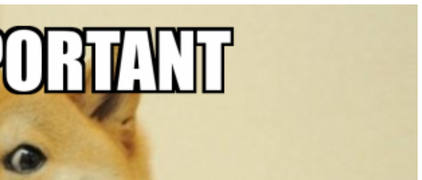

SO IMPORTANT
MUCH WOW
makeameme.org

SO IMPORTANT
MUCH WOW
makeameme.org

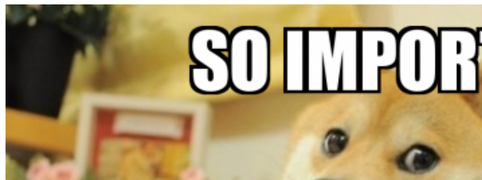
SO IMPORTANT
MUCH WOW
makeameme.org

keameme.org

SO IMPORTANT
MUCH WOW
makeameme.org

SO IMPORTANT
MUCH WOW
makeameme.org

SO IMPORTANT
MUCH WOW
makeameme

ORTANT

SO IMPORTANT

SO IMPORTANT

SO IMPOR

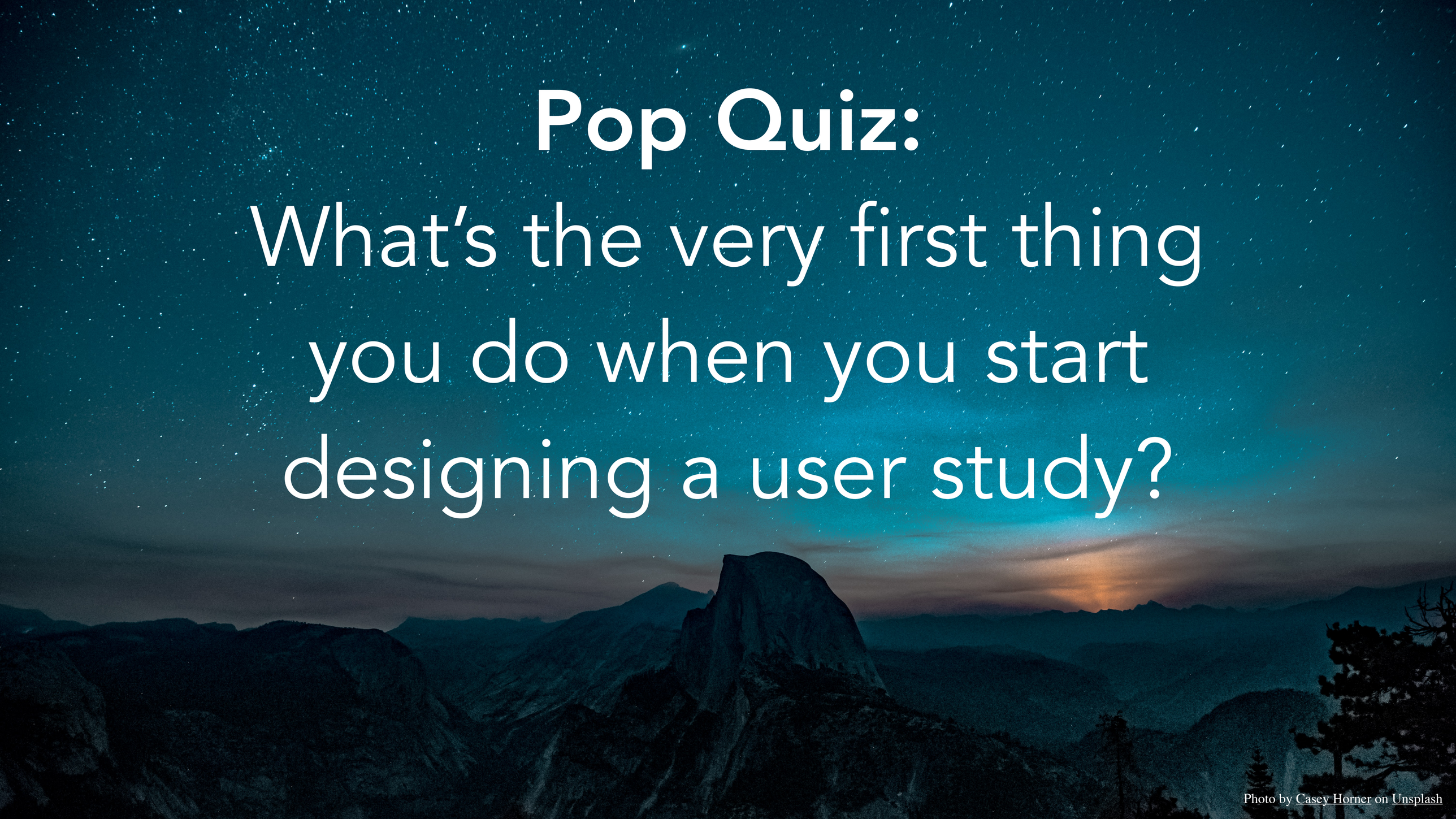


Photo by Casey Horner on Unsplash

What's the very first thing you do when you start designing a user study?

Choose your research question!!!!

Photo by Casey Horner on Unsplash

Why is choosing your research question the first step?

- We’re not observing users just for the sake of observing users!
- We want to learn things that will be useful for our language, environment, or tool design process.
- How can assess whether our study design is helping us answer our key design questions if we haven’t actually *identified our key design questions*?

Three Categories of User Study RQs

Need Finding Study

What are interesting **problems** to solve?

Formative Study

For a given **problem**, what are promising **solutions**?

Evaluative Study

For a given **problem**, now that we've implemented a **solution**, did it work?

Three Categories of User Study RQs

Need Finding Study

What are interesting **problems** to solve?

This week's topic

Formative Study

For a given **problem**, what are promising **solutions**?

Evaluative Study

For a given **problem**, now that we've implemented a **solution**, did it work?

Shape of a Need Finding RQ

What kinds of problems does <description of audience> face during <description of tasks>?

For this class, usually…

What kinds of problems does <description of audience> face during <description of *programming* tasks>?

Grace Mallon
@GraceMallon3

PhD student, c.2020: Here's a limited argument I made based on years of specialized research. Hope it's OK 😕

Philosopher dude, c.1770: Here are some Thoughts I had in the Bath. They constitute Universal & Self-Evident Laws of Nature. FIGHT ME.

Need Finding for PL

COMMUNICATIONS OF THE ACM

HOME | CURRENT ISSUE | NEWS | BLOGS | OPINION | RESEARCH | PRACT

Home / Magazine Archive / August 2021 (Vol. 64, No. 8) / PL and HCI: Better Together / Full Text

REVIEW ARTICLES

PL and HCI: Better Together

By Sarah E. Chasins, Elena L. Glassman, Joshua Sunshine
Communications of the ACM, August 2021, Vol. 64 No. 8, Pages 98-106
10.1145/3469279
Comments

In the last 10 years, the computer science (CS) community has developed novel programming systems that are transforming our world. Data journalists are wielding new programming tools to enrich many major media outlets with interactive visualizations. Microsoft Excel, the primary data programming environment for hundreds of millions of people, now comes with a program synthesis tool that helps users clean and transform their data, sparing them from writing painful spreadsheet formulas. These

“If I had asked people what they wanted, they would have said faster horses.”

- Need finding is ***not*** about asking participants what they want and then doing what they say they want.
- Need finding isn’t even part of the brainstorming process! We’re not deciding what to build or design here. We’re just doing what the name says—finding needs.
  - We’re finding problems. We’ll brainstorm solutions later.
- Good need finding also typically **doesn’t** involve asking people what they want.

Show, Don’t Tell

- We want to structure our need finding interactions so that users show, don’t tell. Why?
  - **We could miss true things.** Users don’t know all their needs! There are some that we could observe that they’d never notice themselves.
  - **We could learn false things.** Memory and introspection unreliable. (Startlingly reliable results in psych.)
  - **We could learn true things poorly.** Easy to come away with a shallow understanding of a need.

- Our **number 1 need finding tool is observation**—just watching participants do their thing
  - Enforces this show-don’t-tell idea very naturally
  - Unless you have very, very good reasons not to do contextual inquiry, I usually recommend starting there!

PL Observation

- Watch a participant using their current programming tools.
  - Where do they struggle or get frustrated?
  - Where do they do things you'd do differently?
  - Where do they have to hop out of their programming environment and look elsewhere or use an extra tool?
  - Where do they have an established workaround for a given issue?
- Give a participant a new programming tool, then look for the same questions. (Especially for looking for learnability needs! Not usually a good fit for other need finding goals.)
- Give a participant similar tasks with multiple programming tools, same questions.
- Attend meetings with participants.
  - I know, I know, boring. But…
    - What concepts, information, data do they pull to mind, express, or draw easily? Which are hard?
    - What goals do they express that they haven't tackled yet. Why?
  - Especially useful for working with non-programmers

Contextual Inquiry for PL

CI is the one where we watch people doing their thing. We ask about their actions when we get confused, when we don't follow. But mostly we're trying to learn about their process. This is wildly useful for PL design.

Contextual Inquiry for PL

Study 1. Observed 13 developers, tasks set by researchers, unfamiliar codebase.
Study 3. Observed 17 developers, developers' own tasks.

Developers Ask Reachability Questions

Thomas D. LaToza
Institute for Software Research
School of Computer Science
Carnegie Mellon University
tlatoza@cs.cmu.edu

Brad A. Myers
Human Computer Interaction Institute
School of Computer Science
Carnegie Mellon University
bam@cs.cmu.edu

ABSTRACT

A *reachability question* is a search across feasible paths through a program for target statements matching search criteria. In three separate studies, we found that reachability questions are common and often time consuming to answer. In the first study, we observed 13 developers in the lab and found that half of the bugs developers inserted were associated with reachability questions. In the second study, 460 professional software developers reported asking questions that may be answered using reachability questions more than 9 times a day, and 82% rated one or more as at least somewhat hard to answer. In the third study, we observed 17 developers in the field and found that 9 of the 10 longest activities were associated with reachability questions. These findings suggest that reachability questions are important to software development.

which in turn caused half of the reported bugs [15]. Successfully coordinating dependencies among effects in loosely connected modules can be very challenging [5].

To better understand how developers understand large, complex codebases, we conducted three studies of developers' questions during coding tasks. Surprisingly, we discovered that a significant portion of developer's work involves answering what we call *reachability questions*. A reachability question is a search across all feasible paths through a program for statements matching search criteria. Reachability questions capture much of how we observed developers reasoning about causality among behaviors in a program.

Contextual Inquiry for PL

A Contextual Inquiry of Expert Programmers in an Event-Based Programming Environment

Amy J. Ko
Human-Computer Interaction Institute
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213 USA
ajko@cmu.edu

Observed 4 developers, completed a total of 12 hours of contextual inquiry (broken into 12 separate sessions). Course-provided tasks.

ABSTRACT
Event-based programming has been studied little, yet recent work suggests that language paradigm can predict programming strategies and performance. A contextual inquiry of four expert programmers using the Alice 3D programming environment was performed in order to discover how event-based programming strategies might be supported in programming environments. Various programming, testing, and debugging breakdowns were extracted from observations and possible programming environment tools are suggested as aids to avoid these breakdowns. Future analyses and studies are described.

METHOD
Participating programmers were enrolled in the "Building Virtual Worlds" course offered at Carnegie Mellon University. The course requires collaborations among programmers, modelers, sound engineers, and painters to create a new interactive 3D world every two weeks using Alice (see Figure 1). Alice provides a limited object model, global event handlers, and a strictly enforced structured editor, preventing all syntax errors.

Four expert programmers were recruited and observed during the second half of the semester, after the programmers were experienced with Alice. Other than Alice, the least expert programmer had experience with 3

Semi-CI Observation for PL

How Statically-Typed Functional Programmers Write Code

JUSTIN LUBIN, University of California, Berkeley, USA
SARAH E. CHASINS, University of California, Berkeley, USA

How working statically-typed functional programmers write code is largely understudied. And yet, a better understanding of developer practices could pave the way for the design of more useful and usable tooling, more ergonomic languages, and more effective on-ramps into programming communities. The goal of this work is to address this knowledge gap: to better understand the high-level authoring patterns that statically-typed functional programmers employ. We conducted a grounded theory analysis of 30 programming sessions of practicing statically-typed functional programmers, 15 of which also included a semi-structured interview. T theory we developed gives insight into how the specific affordances of statically-typed functional programming affect domain modeling, type construction, focusing techniques, exploratory and reasoning strategies, and expressions of intent. We conducted a set of quantitative lab experiments to validate our findings, including that statically-typed functional programmers often iterate between editing types and expressions, that they often run their compiler on code even when they know it will not successfully compile, and that they make textual program edits that reliably signal future edits that they intend to make. Lastly, we outline the implications of our findings for language and tool design. The success of this approach in revealing program authorship patterns suggests that the same methodology could be used to study other understudied programmer populations.

CCS Concepts: • **Human-centered computing** → **HCI theory, concepts and models**; • **Software and its engineering** → **Functional languages**.

Additional Key Words and Phrases: static types, functional programming, grounded theory, need-finding, interviews, qualitative, quantitative, mixed methods, randomized controlled trial

Observed 30 developers, 15 researcher-conducted study sessions (researcher-selected tasks, participant-selected langs) and 15 recorded programming sessions from livestreaming websites.

Semi-CI Observation for PL

Exploring End User Programming Needs in Home Automation

JULIA BRICH, MARCEL WALCH, MICHAEL RIETZLER, and MICHAEL WEBER,
Ulm University
FLORIAN SCHAUB, Ulm University, Carnegie Mellon University, and University of Michigan

Home automation faces the challenge of providing ubiquitous, unobtrusive services while empowering users with approachable configuration interfaces. These interfaces need to provide sufficient expressiveness to support complex automation, and notations need to be devised that enable less tech-savvy users to express such scenarios. Rule-based and process-oriented paradigms have emerged as opposing ends of the spectrum; however, their underlying concepts have not been studied comparatively. We report on a contextual inquiry study in which we collected qualitative data from 18 participants in 12 households on the current potential and acceptance of home automation, as well as explored the respective benefits and drawbacks of these two notation paradigms for end users. Results show that rule-based notations are sufficient for simple automation tasks but not flexible enough for more complex use cases. The resulting insights can inform the design of interfaces for smart homes to enable usable real-world home automation for end users.

CCS Concepts: • **Human-centered computing** → **Empirical studies in interaction design**; *Ubiquitous and mobile computing systems and tools*; • **Software and its engineering** → Software notations and tools;

Additional Key Words and Phrases: Configuration interfaces, contextual inquiry, qualitative analysis, smart home

ACM Reference Format:
Julia Brich, Marcel Walch, Michael Rietzler, Michael Weber, and Florian Schaub. 2017. Exploring end user

18 participants, 12 households. Home tour (!!) followed by a think-aloud study using one of two home automation programming paradigms. (Researcher-assigned tasks.)

11

How else can we observe in PL contexts?

Non-CI Observation for PL

L. LOUCA[1], A. DRUIN, D. HAMMER, D. DREHER

STUDENTS' COLLABORATIVE USE OF COMPUTER-BASED PROGRAMMING TOOLS IN SCIENCE: A DESCRIPTIVE STUDY

Submitted to CSCL Conference 2003

Abstract: This paper presents a small-scale study investigating the use of two different computer-based programming environments (CPEs) as modeling tools for collaborative science learning with fifth grade students. We analyze student work and conversations while working with CPEs using *Contextual Inquiry*. Findings highlight the differences in activity patterns between groups using different CPEs. Students using *Stagecast Creator (SC)* did twice as much planning but half as much debugging compared with students using *Microworlds (MW)*. Students working with MW were using written code on the computer screen to communicate their ideas whereas students working with SC were using the programming language to talk about their ideas prior to any programming. We propose three areas for future research. (1) Exploring different types of communication styles as compared with the use of different CPEs. (2) Identifying students' nascent abilities for using CPEs to show functionality in science. (3) Further understanding CPEs' design characteristics as to which may promote or hamper learning with models in science.

Observed student users of two different programming tools, identified differences in how they spent their time. Observed 9 5th graders in science class. Not previously familiar with the programming environments. 10 meetings of 45-60 minutes with the whole group. Students split into 3 groups of 3 to work with the programming tools.

Non-CI Observation for PL

How Should Compilers Explain Problems to Developers?

Titus Barik
Microsoft
Redmond, WA, USA
titus.barik@microsoft.com

Denae Ford
NC State University
Raleigh, NC, USA
dford3@ncsu.edu

Emerson Murphy-Hill
NC State University
Raleigh, NC, USA
emerson@csc.ncsu.edu

Chris Parnin
NC State University
Raleigh, NC, USA
cjparnin@ncsu.edu

ABSTRACT

Compilers primarily give feedback about problems to developers through the use of error messages. Unfortunately, developers routinely find these messages to be confusing and unhelpful. In this paper, we postulate that because error messages present poor explanations, theories of explanation—such as Toulmin's model of argument—can be applied to improve their quality. To understand how compilers should present explanations to developers, we conducted a comparative evaluation with 68 professional software developers and an empirical study of compiler error messages found in Stack Overflow questions across seven different programming languages.

Our findings suggest that, given a pair of error messages, developers significantly prefer the error message that employs proper argument structure over a deficient argument structure when neither offers a resolution—but will accept a deficient argument structure if it provides a resolution to the problem. Human-authored explanations on Stack Overflow converge to one of the three argu-

1 INTRODUCTION

Compilers primarily give feedback about problems to developers through the use of error messages.[1] Despite the intended utility of error messages, researchers and practitioners alike have described their output as "cryptic" [44], "difficult to resolve" [44], "not very helpful" [48], "appalling" [5], "unnatural" [6], and "basically impenetrable" [40].

While poor error messages are paralyzing for novices, even experienced developers have substantial difficulties when comprehending and resolving them. A study conducted at Google found that nearly 30% of builds fail due to a compiler error, and that the median resolution time for each error is 12 minutes [38]. Surprisingly, the costly errors that developers make are rather mundane, relating to basic issues such as dependencies, type mismatches, syntax, and semantic errors. Barik et al. [2] conducted an eye-tracking study with developers and found that they spent up to 25% of their task time on reading error messages. In addition, developers in a study by Johnson et al. [19] reported that error messages were often not useful because they did not adequately *explain* the problem.

Stack Overflow is a record of real questions and confusions that programmers encounter in their practice. Votes on answers offer evidence of what kinds of responses are helpful to them. This is kind of a log of observations! How can we use this info to improve compiler error messages, which also offer feedback when programming tasks go wrong?

Non-CI Observation for PL

GitHub might not be a log of actual user behavior, but at least it's a log of the programs they end up with…

An Empirical Study of Goto in C Code from GitHub Repositories

Meiyappan Nagappan[1], Romain Robbes[2], Yasutaka Kamei[3], Éric Tanter[2],
Shane McIntosh[4], Audris Mockus[5], Ahmed E. Hassan[6]
[1]Rochester Institute of Technology, Rochester, NY, USA; [2]Computer Science Department (DCC), University of Chile, Santiago, Chile; [3]Kyushu University, Nishi-ku, Japan; [4]McGill University, Montreal, Canada; [5]University of Tennessee-Knoxville, Knoxville, Tennessee, USA; [6]Queen's University, Kingston, Ontario, Canada
[1]mei@se.rit.edu, [2]{rrobbes, etanter}@dcc.uchile.cl, [3]kamei@ait.kyushu-u.ac.jp, [4]shanemcintosh@acm.org, [5]audris@utk.edu, [6]ahmed@cs.queensu.ca

ABSTRACT

It is nearly 50 years since Dijkstra argued that `goto` obscures the flow of control in program execution and urged programmers to abandon the `goto` statement. While past research has shown that `goto` is still in use, little is known about whether `goto` is used in the unrestricted manner that Dijkstra feared, and if it is 'harmful' enough to be a part of a post-release bug. We, therefore, conduct a two part empirical study - (1) qualitatively analyze a statistically representative sample of 384 files from a population of almost 250K C programming language files collected from over 11K GitHub repositories and find that developers use `goto` in C files for error handling (80.21±5%) and cleaning up resources at the end of a procedure (40.36 ± 5%); and (2) quantitatively analyze the commit history from the release branches of six OSS projects and find that no `goto` statement was removed/modified in the post-release phase of four of the six projects. We conclude that developers limit themselves to using `goto` appropriately in most cases, and not in an unrestricted manner like Dijkstra feared, thus suggesting that `goto` does not appear to be harmful in practice.

Harmful [11]. This is one of the many works of Dijkstra that is frequently discussed by software practitioners [25] and researchers alike (more than 1,300 citations according to Google Scholar and almost 4000 citations according to ACM Digital Library as of Aug 15, 2014). This article has also resulted in a slew of other articles of the type *global variables considered harmful* [32], *polymorphism considered harmful* [24], *fragmentation considered harmful* [16], among many others. In fact, Meyer claims that as of 2002, there are thousands of such articles, though most are not peer-reviewed [15].

Indeed, Dijkstra's article [11] has had a tremendous impact. Anecdotally, several introductory programming courses instruct students to avoid `goto` statements solely based on Dijkstra's advice. Marshall and Webber [19] warn that when programming constructs like `goto` are forbidden for long enough, they become difficult to recall when required.

Dijkstra's article on the use of `goto` is based on his desire to make programs verifiable. The article is not just an opinion piece; as Koenig points out [7], Dijkstra provides strong *logical* evidence for why `goto` statements can intro-

How else might we observe people programming to find needs?

- In-lab observation, observation with assigned tasks as opposed to users' own
- Found logs—stackoverflow, github, so on
- You can instrument a programming environment to log various user actions
  - But don't be creepy! (Easy to get intrusive with tracking)
- In a course context, you can instrument the automatic test infrastructure, if applicable
- These days people stream themselves programming! You can watch those
- More ideas? Raise hand.

Show, Don’t Tell..the next best thing

- If you really can’t manage contextual inquiry, can you set up another way to do observation?
- Ok, if you *really* can’t manage observation, what next?
- Get concrete. It gets us closer to “showing"
  - ~~“What’s hard about programming for you?”~~
  - “In your most recent programming project, what was the most frustrating part? Can you walk me through how it came up? Why it was frustrating? How you ultimately dealt with it?”
- Get open-ended. Yes/No answers don’t give us a lot. Stories give us much more.
  - ~~“Do you prefer Python or R?”~~
  - “Have you found that some programming tasks are much easier in different programming languages? Can you tell me about the last time you found one of these and how?”

Alternatives to Contextual Inquiry for PL

Semi-structured interview to identify possible issues in the programming process, followed by survey to collect quantitative evidence of issues uncovered in interviews.

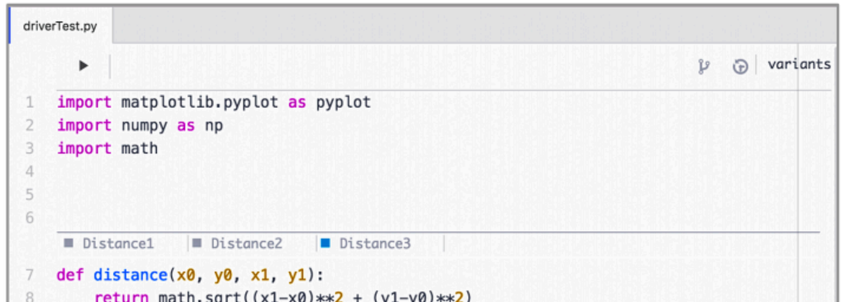
Variolite: Supporting Exploratory Programming by Data Scientists

Mary Beth Kery
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA
mkery@cs.cmu.edu

Amber Horvath
Oregon State University
Corvallis, Oregon, USA
horvatha@oregonstate.edu

Brad Myers
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA
bam@cs.cmu.edu

ABSTRACT
How do people ideate through code? Using semi-structured interviews and a survey, we studied data scientists who program, often with small scripts, to experiment with data. These studies show that data scientists frequently code new analysis ideas by building off of their code from a previous idea. They often rely on informal versioning interactions like copying code, keeping unused code, and commenting



Alternatives to Contextual Inquiry for PL

Interviews

Enabling Data-Driven API Design with Community Usage Data: A Need-Finding Study

Tianyi Zhang[†], **Björn Hartmann**[§], **Miryung Kim**[‖], **Elena L. Glassman**[†]
[†]Harvard University, MA, USA
[§]UC Berkeley, Berkeley, CA, USA
[‖]UC Los Angeles, Los Angeles, CA, USA
{tianyi, eglassman}@seas.harvard.edu, bjoern@berkeley.edu, miryung@cs.ucla.edu

ABSTRACT
APIs are becoming the fundamental building block of modern software and their usability is crucial to programming efficiency and software quality. Yet API designers find it hard to gather and interpret user feedback on their APIs. To close the gap, we interviewed 23 API designers from 6 companies and 11 open-source projects to understand their practices and needs. The primary way of gathering user feedback is through bug reports and peer reviews, as formal usability testing is prohibitively expensive to conduct in practice. Participants

tions to computers.[1] The use of APIs is ubiquitous, powering software applications, systems, and web services in nearly every domain. Given the increasing number and complexity of APIs, learning and using APIs is becoming a common activity and a key challenge in modern programming [27, 47, 48, 40].

User-centered design can produce usable APIs with great clarity, learnability, and programming efficiency [40, 39, 52]. Traditional usability testing methods such as user studies are often deemed too expensive to conduct during API design [19]. For

Alternatives to Contextual Inquiry for PL

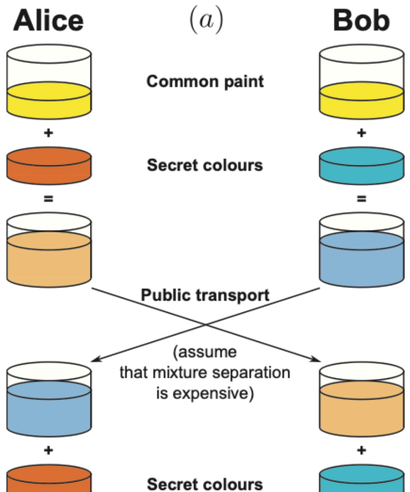
Interviews

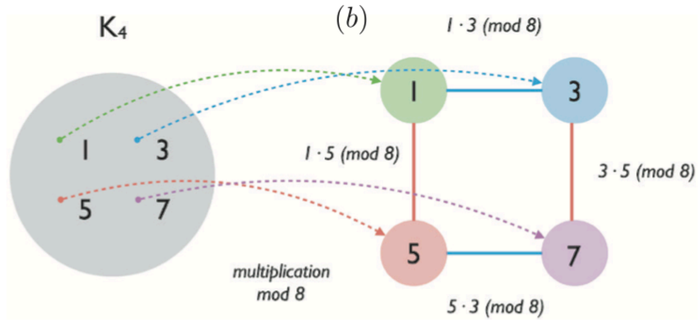
How Domain Experts Create Conceptual Diagrams and Implications for Tool Design

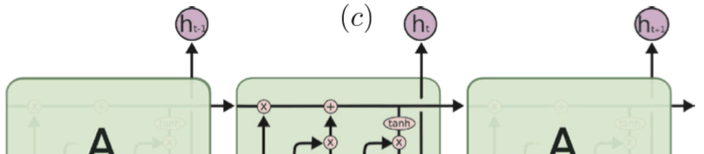
Dor Ma'ayan[*1,2] **Wode Ni**[*2] **Katherine Ye**[2] **Chinmay Kulkarni**[2] **Joshua Sunshine**[2]

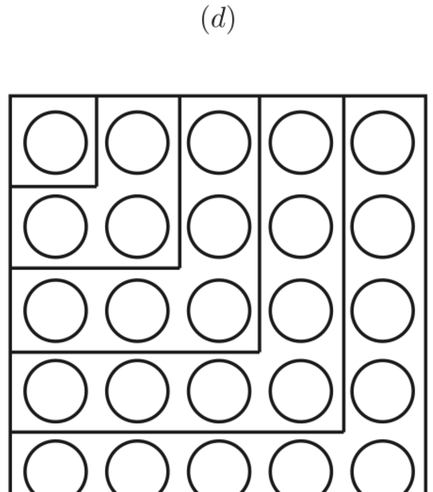
[1]Technion - Israel Institute of Technology
Haifa, Israel
dorma10@campus.technion.ac.il

[2]Carnegie Mellon University
Pittsburgh, PA
woden, kqy, chinmayk, sunshine@cs.cmu.edu



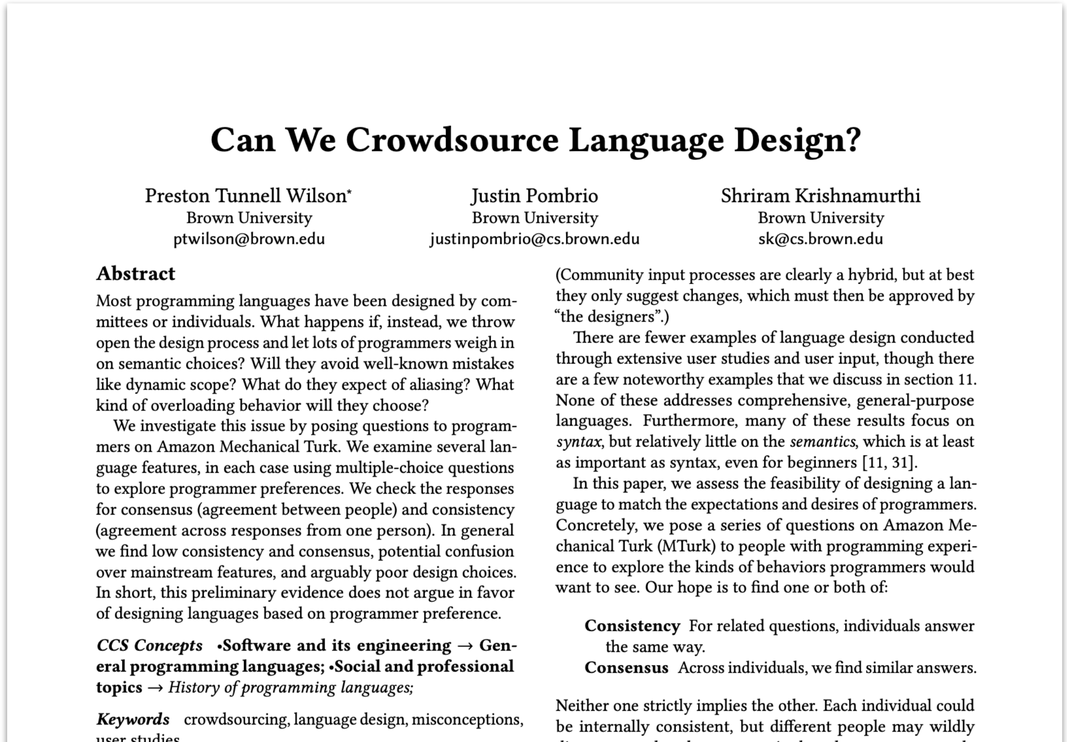






Show, Don’t Tell

- Surveys — are they out?
  - No! But we have to find ways to get them to “show” via the survey.
    - Don’t ask how often they use construct A, ask them to upload their last program so you can count uses of A



Can We Crowdsource Language Design?

Preston Tunnell Wilson*
Brown University
ptwilson@brown.edu

Justin Pombrio
Brown University
justinpombrio@cs.brown.edu

Shriram Krishnamurthi
Brown University
sk@cs.brown.edu

Abstract

Most programming languages have been designed by committees or individuals. What happens if, instead, we throw open the design process and let lots of programmers weigh in on semantic choices? Will they avoid well-known mistakes like dynamic scope? What do they expect of aliasing? What kind of overloading behavior will they choose?

We investigate this issue by posing questions to programmers on Amazon Mechanical Turk. We examine several language features, in each case using multiple-choice questions to explore programmer preferences. We check the responses for consensus (agreement between people) and consistency (agreement across responses from one person). In general we find low consistency and consensus, potential confusion over mainstream features, and arguably poor design choices. In short, this preliminary evidence does not argue in favor of designing languages based on programmer preference.

CCS Concepts **•Software and its engineering → General programming languages; •Social and professional topics** → *History of programming languages;*

Keywords crowdsourcing, language design, misconceptions, user studies

(Community input processes are clearly a hybrid, but at best they only suggest changes, which must then be approved by “the designers”.)

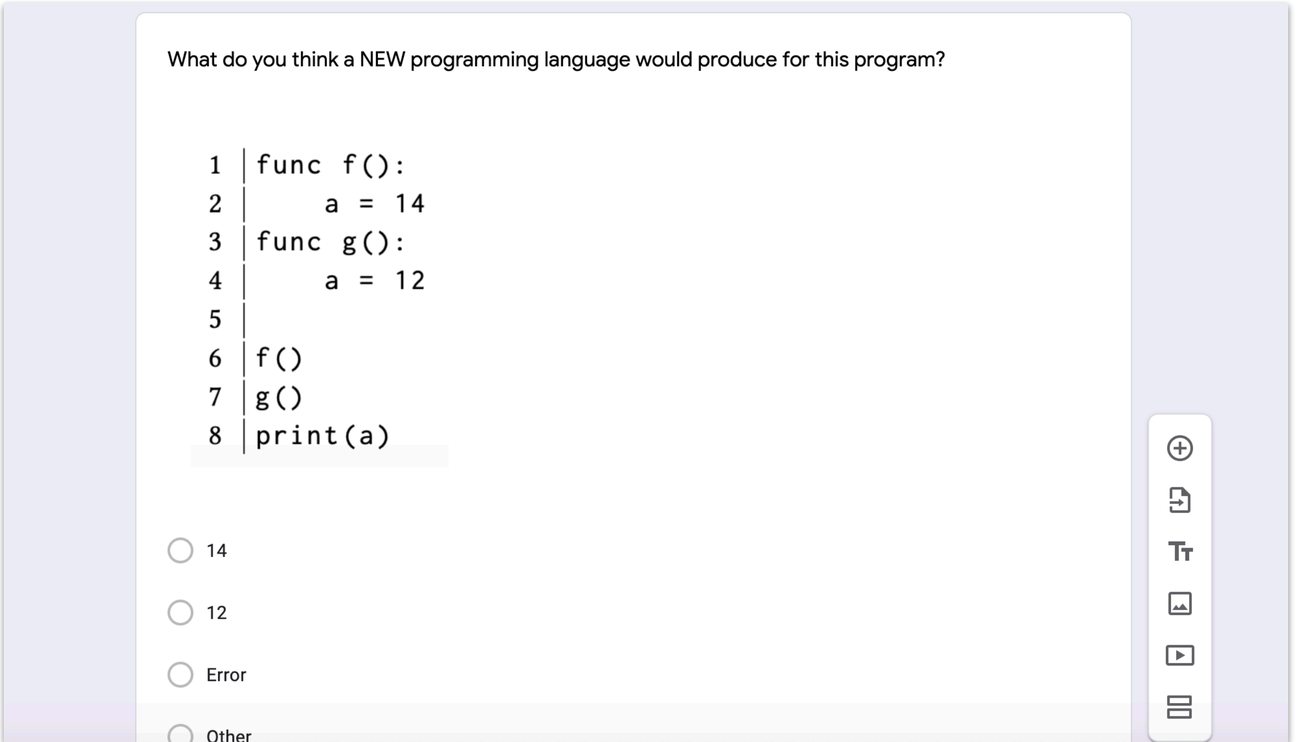
There are fewer examples of language design conducted through extensive user studies and user input, though there are a few noteworthy examples that we discuss in section 11. None of these addresses comprehensive, general-purpose languages. Furthermore, many of these results focus on *syntax*, but relatively little on the *semantics*, which is at least as important as syntax, even for beginners [11, 31].

In this paper, we assess the feasibility of designing a language to match the expectations and desires of programmers. Concretely, we pose a series of questions on Amazon Mechanical Turk (MTurk) to people with programming experience to explore the kinds of behaviors programmers would want to see. Our hope is to find one or both of:

Consistency For related questions, individuals answer the same way.

Consensus Across individuals, we find similar answers.

Neither one strictly implies the other. Each individual could be internally consistent, but different people may wildly



Show, Don’t Tell

Research Question: Are there gaps between program semantics and programmer expectations about semantics?

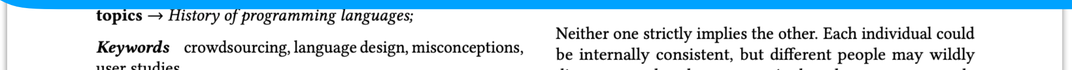
Tell version. “Describe some language features that you find surprising.”
Tell version. “Do you expect a new programming language to have static or dynamic scope?”
Show version. “What output do you expect here?” “And here?”

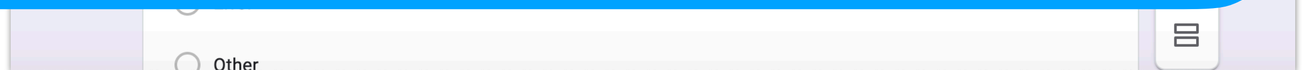
Outcome: Programmers weren’t consistent! In one program (survey question) they’d give answer consistent with static scope, in another with dynamic scope.

Is this successful need finding? Yes! We didn’t find a solution—we can’t say ok, use static scope and programmers won’t be surprised anymore. But that’s not the goal! The goal is to find problems, not solutions.

Goal isn’t even to find out what programmers want, even though the questions may make it look like that. (Remember, asking is a bad way to figure that out…) It was to learn about mismatches between semantics and expectations, and by finding programmer inconsistency they found mismatches.

And this inconsistency is another reason we don’t just ask what people want. :)





You *can* ask "would" questions…but be careful

This question is my number 1 trick for getting to useful conversations in discussions with social scientists when I don't have time for a full contextual inquiry process with them!

- Audience matters
  - If you're working with novice programmers or non-programmers…
    - ~~"What would you like to automate that you don't automate right now?"~~
    - "What would you do if you had 100 interns for the next three months?"
- And programmers aren't great at "would" questions either…
  - ~~"What would make this programming environment better?"~~
    - "This menu is in a bad place, this font is too small, this pane should be on the other side…"
    - It's not that no one should be collecting this feedback or that we shouldn't solve problems like these. But if you're in this class, I suspect this isn't the class of user input you're seeking!
- And remember that these questions are for revealing hopes and dreams, don't necessarily reflect how they'd actually act
  - But difference between actions and hopes/dreams can be revealing!

Assignment 2: Show, Don’t Tell

- Assignment 2
  - Ideally, go out in the world and watch people do their work in context! If not possible, watch their work on Zoom. Don’t let the design of Assignment 2 make you think an interview is a substitute for that process
  - During the Assignment 2 work time, see if you can find a way to make your session not about an interview but about watching them do their work/hobby/task that you want to study, with occasional interruptions for you to learn about what they’re doing.
    - If their task is computer-based, can they screen share?
    - If their task is non-computer based, can they point the camera at it?
  - Suggested structure:
    - Describe the kinds of tasks you’re interested in observing.
    - Ask the participant to teach/show you how they do those tasks. Interrupt when something happens that you don’t understand.
    - In the last 10 or 15 minutes, run your observations by the participant to see what you got right or wrong about their process.
  - Also highly encourage reading Thursday’s reading before finalizing your design!