Types of Studies

Brad A. Myers

Michael Coblenz, Jonathan Aldrich, Joshua Sunshine

Human-Computer Interaction Institute
School of Computer Science
Carnegie Mellon University

bam@cs.cmu.edu
bradamyers.com

Dagstuhl Seminar 18061
Evidence About Programmers for Programming Language Design

© 2018 – Brad A. Myers

Talk Based on:

- Michael Coblenz, Jonathan Aldrich, Joshua Sunshine, Brad Myers, *Interdisciplinary Programming Language Design.* (draft distributed here)
  - *Comments requested!*

- Brad A. Myers, Andrew J. Ko, Thomas D. LaToza, and YoungSeok Yoon. "Programmers Are Users Too: Human-Centered Methods for Improving Programming Tools," *IEEE Computer*, Special issue on UI Design, 49, issue 7, July, 2016, pp. 44-52. IEEE DL or local pdf

© 2018 – Brad A. Myers

“Design”

- Much more to the “design” of a language than just *evaluation*
  - Design is a creative process
- Many methods which can be used at different parts of the process
  - That answer different questions
- Different people will have different perspectives and goals

SCHLOSS DAGSTUHL
Leibniz-Zentrum für Informatik

About Dagstuhl | Program | Public

You are here: Program » Seminar Calendar » Seminar Homepage

Seminars
Perspectives
ıhl Seminars
Guests
Calendar
ıts

http://www.dagstuhl.de/18061

February 4 – 9 , 2018, Dagstuhl Seminar 18061

Evidence About Programmers for Programming Language Design

Organizers

Stefan Hanenberg (Universität Duisburg-Essen, DE)

© 2018 – Brad A. Myers

Language vs. Environment vs. APIs

- This meeting focuses on the *programming language* itself
- For some languages, not really separable from the editor or IDE
  - E.g., Visual Programming languages
- Fluid about what is in the "language" vs. in its libraries (APIs)
  - I/O, Multi-processing, Glacier's immutability [Coblenz, ICSE'17]
- But still focusing on language itself

SCHLOSS DAGSTUHL
Leibniz-Zentrum für Informatik

About Dagstuhl | Program | Public

You are here: Program » Seminar Calendar » Seminar Homepage

Seminars
Perspectives
ıhl Seminars
Guests
Calendar
ıts

http://www.dagstuhl.de/18061

February 4 – 9 , 2018, Dagstuhl Seminar 18061

Evidence About Programmers for Programming Language Design

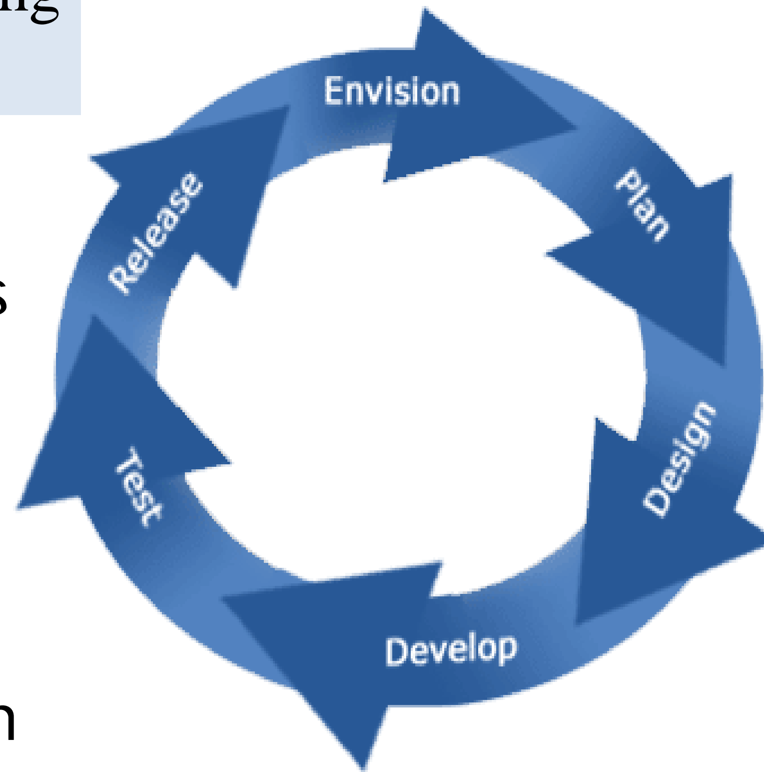
Organizers

Stefan Hanenberg (Universität Duisburg-Essen, DE)

Many HCI Methods for improving programmer productivity

Cite: Brad A. Myers, Andrew J. Ko, Thomas D. LaToza, and YoungSeok Yoon. "Programmers Are Users Too: Human-Centered Methods for Improving Programming Tools," *IEEE Computer*, Special issue on UI Design, 49, issue 7, July, 2016, pp. 44-52. IEEE DL or local pdf

We have used a variety of HCI methods to improve programming tools across the lifecycle.



Exploratory Studies

- Contextual Inquiries
- Interviews
- Surveys
- Lab Studies
- Corpus data mining

Field Studies

- Logs & error reports

Evaluative Studies

- Expert analyses
- Usability Evaluation
- Formal A/B Lab Testing

Design Practices

- "Natural programming"
- Graphic & Interaction Design
- Prototyping

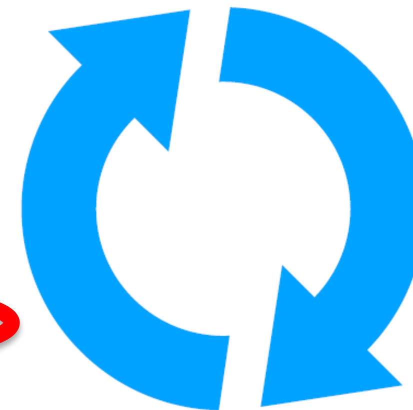
© 2018 – Brad A. Myers

Simpler Model in New Paper

- Combined needs-finding, design, implementation
- Combined all parts of evaluation
- Some methods are not "human-centered"
- Still iterative
- All are "design"
- We are the "designers"
  - vs. users of languages
- Both are "programmers"

Evaluation

Performance evaluation
User experiments
Case studies
Expert evaluation
Formalism and proof
Qualitative user studies



Requirements and Creation

Interviews
Corpus studies
Natural Programming
Rapid Prototyping

© 2018 – Brad A. Myers

Overview

- Desiderata of Programming Languages (Goals)
- Perspectives on Language Design (Roles)
- Methods for Design
- Interdisciplinary Approach
- (If time) Examples of our use of these methods

© 2018 – Brad A. Myers

Desiderata of Programming Languages (Goals)

- What designer wants language to be good for
- Based on software engineering *quality attributes*
  - Most can be affected by programming language design
- Design always involves *tradeoffs* among the attributes
- Must decide which goals to focus on
  - What should be optimized?
    - Depends on the designer's aims for the language or feature
  - What evaluation methods are appropriate?
    - Learnability measured differently than Correctness
- Must be intentional and explicit about priorities

© 2018 – Brad A. Myers

Goals (traditional)

1. Correctness
   - Adherence to a specification
   - Absence of bugs
   - **Supported by**: type and proof systems
   - **Evaluated by**: proofs of soundness theorems

2. Performance
   - *Of resulting code* (not of the programmer or compiler)
   - **Evaluated by**: benchmarks

Goals (traditional)

3. Expressiveness
   - Programmers can express their intent explicitly
   - May trade off with usability
     - Programmers have to express more
     - Modifiability – better if can be checked, but more work to change
   - **Supported by**: type systems, domain-specific features
   - **Evaluated by**: case studies, examples
4. Speed of Compiling
   - Has always been a concern (e.g., C is one-pass; Go modules)
   - **Evaluated by**: measuring compile time of benchmark code

© 2018 – Brad A. Myers

Goals (usability)

5. Understandability
   - Readability
   - 10:1 ratio reading vs. writing
   - Considers both: what does *this* code do & *where* is the code that does *X*
   - **Supported by**: appropriate keywords, syntax, and features
   - **Evaluated by**:
     - User studies of reading code
     - Can programmers answer important questions
6. Ease of Reasoning
   - User-focused analog of correctness
   - Should be user-centered, not just designer's
   - **Supported by**: modules (separation); concise proofs of correctness
   - **Evaluated by:** user studies

© 2018 – Brad A. Myers

Goals (usability)

7. Modifiability
   - Ease of making changes to the code (ref: cog. dim.'s *viscosity*)
   - Key software engineering requirement
   - **Supported by**: information hiding
   - **Evaluated by**: user studies of editing; case studies for larger modifications
8. Learnability
   - Key to adoption
   - Key requirement for schools
   - **Supported by**: fewer concepts (e.g., removing textual syntax), good pedagogy, being similar to known languages
   - **Evaluated by:** lab or classroom studies with novices

© 2018 – Brad A. Myers

Goals (Discussion)

- Other goals that are not listed?
- I proposed “speed of entering code”?
- What about “scalability”?
  - Multiple people writing code for large systems
- “Debuggability”?
- Other quality attributes or usability attributes not covered?

© 2018 – Brad A. Myers

Perspectives on Language Design (Roles)

- Kinds of people – perspectives
  - Affects what they *value* in the design
  - Which of the goals are most important
- Not researcher vs. practitioner
- Most people combine multiple roles
- It was hard to pick good names for these
  - Trying to be non-judgmental

© 2018 – Brad A. Myers

Roles

1. Logician
   - **Key Focus:** on *correctness;* formal methods
   - Programming by highly-trained experts
   - Programming is a mathematical pursuit
     - *Closeness of mapping* to mathematical thinking
   - **Key Research**: new mathematical principles, e.g., type theory
2. Industrialist
   - Creating new language for large-scale use in companies
   - **Key Focus:** on *performance* and *adoption*
   - *Learnability*
   - *Scalability*

© 2018 – Brad A. Myers

Roles

3. Empiricist
   - **Key Focus:** experiments about specific design decisions
   - Hope to enlighten many aspects of design with human-centered data
   - Programming is a human pursuit
     - *Closeness of mapping* for "regular" people/programmers
   - **Key Research**: human-centered studies
4. Teacher
   - **Key Focus:** *learnability*
   - Avoid irrelevant struggles (e.g., syntax for beginners)
   - Often significant focus on programming environment
   - Rarely cares about scalability, efficiency, etc.
   - If advanced class, may want commercial tools
   - **Key Research**: pedagogy

© 2018 – Brad A. Myers

Roles (discussion)

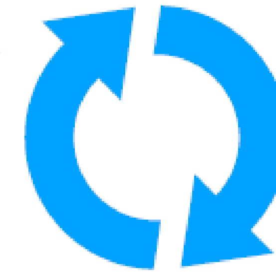
- Do we need other roles to cover the perspectives / desires for languages?
- Are there better names?

© 2018 – Brad A. Myers

Methods

Evaluation

Performance evaluation
User experiments
Case studies
Expert evaluation
Formalism and proof
Qualitative user studies



Requirements and Creation

Interviews
Corpus studies
Natural Programming
Rapid Prototyping

- Requirements and creation
  - Not necessarily have a prototype yet
- Evaluation
  - Have a design
  - At least a prototype
- Some methods may be used for many kinds of information gathering
- My previous talks and papers have shown how my group has used many of these

© 2018 – Brad A. Myers

Methods (Requirements and Creation)

1. Interviews
   - Understand the experience of experts
   - Identify important problems to solve & existing approaches
   - Limited to small numbers
   - Opinions – limited by what is *salient*
2. Surveys
   - Good for identifying how *widespread* a problem is
     - How *important* to address
   - Also opinions; data can be noisy
     - But *not* particularly useful to ask what people *like* best
   - Limited by demographics of respondents

© 2018 – Brad A. Myers

Methods (Requirements and Creation)

3. Corpus Studies
    - Look for patterns in existing code
    - Need hypotheses about what to look for
    - Need a representative corpus
        - Open source may not match closed source
    - (Also can be used for field studies of new designs)
4. Natural Programming
    - A *participatory design* method
    - Elicit how people express solutions without special training
    - *Closeness of mapping; learnability*
    - Limited by participant's prior experiences

Methods (Requirements and Creation)

5. Rapid Prototyping
   - Ubiquitously used in other areas of HCI
   - Low-fi ("paper") prototypes useful to try out early ideas
     - From overall concepts to low-level syntax issues
   - Experimenter plays compiler
     - But hard to do accurately
   - Best to rely on *results* rather than *opinions*
6. PL and SE Theory
   - SE theory characterizes the engineering practices that languages should support (e.g. separate development on different modules)
   - PL theory provides general principles for language design (e.g. distinguishing types and values)
   - PL theory provides a set of well-understood solutions to common language design problems (e.g. memory safety as a way of making a language more secure; object-oriented dispatch as a way of providing extensibility)
   - *(See Jonathan's talk)*

© 2018 – Brad A. Myers

Methods (Evaluation)

7. Qualitative user studies
   - Usability analyses
     - Not numerical or comparative
     - Identifies obstacles and barriers
   - Test feasibility, understandability, learnability
   - (Can also be used on *existing* languages and tools as a *formative* tool – what are the problems)
   - Lab study with direct observations
   - Limited to small tasks, small numbers of users

© 2018 – Brad A. Myers

Methods (Evaluation)

8. Case studies
   - Language designer writes example code
   - Show *expressiveness* and *conciseness*
   - Often targets what a reader might wonder if feasible
   - Sometimes compared to the solution in a different language
   - Limited to a few small cases
   - Only shows that the designer can use it

© 2018 – Brad A. Myers

Methods (Evaluation)

9. Expert evaluation
   - “Cognitive dimensions”, “heuristic analysis,” “cognitive walkthroughs”
   - Good vocabulary for discussing tradeoffs
   - Widely used for VPs, etc.
   - But “just” the evaluator’s opinion
     - Often the designer’s opinion
   - Not validated that correlates with quantitative experiments

© 2018 – Brad A. Myers

Methods (Evaluation)

10. Performance evaluation
    - Typically using speed of resulting code on benchmarks
      - Sometimes standardized
    - SIGPLAN's "empirical evaluations" are *just* for performance ☹
    - Benchmarks may not match real code
    - Real programmers may not be able to write code that has optimal properties

© 2018 – Brad A. Myers

Methods (Evaluation)

11. User experiments
    - Also called: Formal User Studies or randomized controlled trials (RCTs) or A/B Studies
        - Are these all the same?
        - What are "non-randomized, fully controlled experiments"? – Ko/Kaijanaho
    - Show that A has an actual, measurable advantage over B
    - "Gold standard" for academic papers
    - Limited to the specific situation studied
    - Not clear that results of multiple experiments can be *combined*
        - Interaction among features, e.g., lack of consistency

© 2018 – Brad A. Myers

Methods (Evaluation)

12. Formalism and proof

- Show that a language has certain properties, like type soundness
- Used to develop and extend PL theory
- Aids the conceptual integrity of the language design
  - Proof forces designer to really think through the design
- May provide specification and safety guarantees
- Formal verification with tools like Dafny or Coq
- May be a gap between what *can be* specified and what programmers want to do

© 2018 – Brad A. Myers

Methods (discussion)

- What other methods should be included?
  - Eye trackers? – or is that part of usability, experiments?
  - Ko/Kaijanaho: "program pair analyses"?
- HCI teaches > 30 UX methods, what others have people used for PL design?
- What about other methods from other fields?
  - What can / should we "borrow"?
- Do we need to **invent new** methods? For what?

© 2018 – Brad A. Myers

Argue for Interdisciplinary Approach

- Designers should use *multiple* methods from multiple fields
  - Provide complementary kinds of information / evidence
    - One method can address the shortcomings of another
    - Through *triangulation* the whole can be greater than the sum of the individual methods
  - Use methods at all phases of the process
  - "Mixed methods"
- "Successful" languages meet multiple goals
- Be *strategic* in selecting methods
  - What are the questions / claims?
- CMU's collaboration PL/SE + HCI has worked well
  - Myers, Aldrich, Shaw, Herbsleb
  - ~20 PhD students

© 2018 – Brad A. Myers

Argue Against

- We should argue against:
  - × Unsubstantiated claims
  - × Unstated assumptions about what is "best"
    - Must list project/language goals
  - × Use of inappropriate methods
  - × Performing the methods incorrectly
  - × Assuming only one method is valid to use
    - E.g., Formal Methods or Performance or RCTs
  - × Just assuming the conventional wisdom
    - Studies often show it doesn't hold *(see next talk!)*
  - × Inadequate reporting of results
    - × Non-reproducibility of studies – insufficiently documented

© 2018 – Brad A. Myers

Examples

- (If time)
  - We have used a variety of these methods in many projects
- (If not time), See:
  - Our position paper for this meeting
  - Brad A. Myers, Andrew J. Ko, Thomas D. LaToza, and YoungSeok Yoon. "Programmers Are Users Too: Human-Centered Methods for Improving Programming Tools," *IEEE Computer*, Special issue on UI Design, 49, issue 7, July, 2016, pp. 44-52. IEEE DL or local pdf
  - My presentation at Dagstuhl 15222: Brad A. Myers, "Using the Natural Programming Approach Throughout the Lifecycle," *Dagstuhl Conference on Human-Centric Development of Software Tools*, May 25 – 28, 2015, Dagstuhl Seminar 15222, p. 128. pdf. DOI: 10.4230/DagRep.5.5.115

© 2018 – Brad A. Myers

Many User Centered Methods

- Contextual Inquiry
- Contextual Analysis
- Paper prototypes
- Think-aloud protocols
- Heuristic Evaluation
- Affinity diagrams
- Personas
- Wizard of Oz
- Task analysis
- A/B testing
- Cognitive Walkthrough
- Cognitive Dimensions
- KLM and GOMS (CogTool)
- Video prototyping
- Body storming
- Expert interviews
- Questionnaires
- Surveys
- Interaction Relabeling
- Log analysis
- Storyboards
- Focus groups
- Card sorting
- Diary studies
- Improvisation
- Use cases
- Scenarios
- "Speed Dating"
- ...

© 2018 – Brad A. Myers

Example of Interviews: Immutability

- Experts recommend making classes *immutable* so instances cannot change accidentally
  - Thread safe, more secure, no unexpected state changes, etc.
- Usability studies suggest programmers prefer classes that can change
- Various relevant language features
  - C++ `const`, Java `final`, Obj-C immutable collections, .NET `Freezable`, etc.
- Semi-structured interviews with a convenience sample of 8 software engineers
  - Agreed that mutability is a frequent source of bugs
  - But *none* of these features are what is needed
  - Preferred *transitive, class-based immutability*
    - Provided this in the **Glacier** tool *(Coblenz, et. al. ICSE'2016 and ICSE'2017)*
    - **G**reat **L**anguages **A**llow **C**lass **I**mmutability **E**nforced **R**eadily

© 2018 – Brad A. Myers

Corpus Data Mining

- Studied 11 million Java try/catch blocks from GitHub using the Boa tool
- 12% of catch blocks were completely empty.
- 25% of all exceptions caught are simply `Exception`
- Motivated a new tool to help programmers write better exception handling code

[Kery, Le Goues, & Myers, MSR'2016]
[Kistner, Kery, Puskas, Moore & Myers, VL/HCC2017]

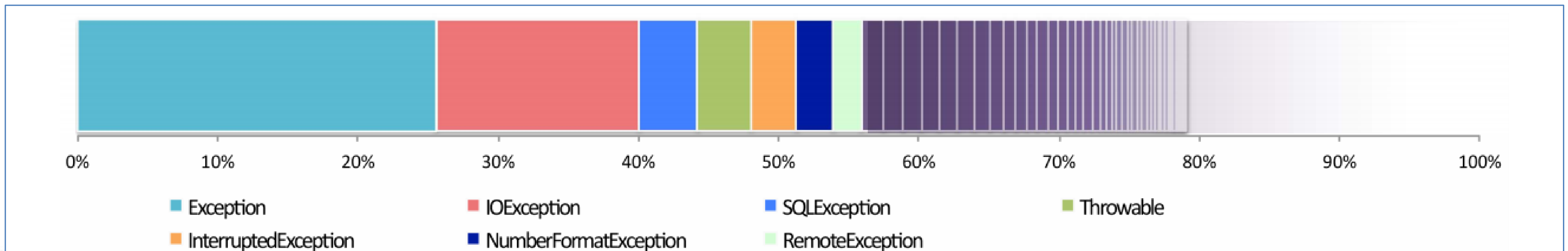
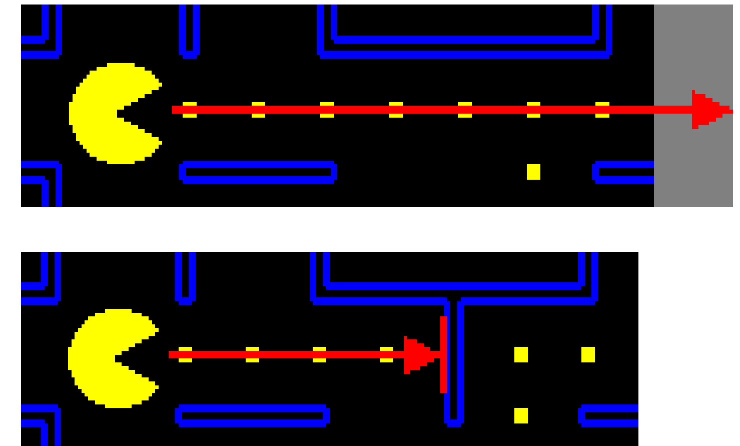


Figure 3: Exceptions caught by catch blocks on GitHub. Exceptions that occur more than 1% of the time are labeled. The rest, in purple, are thousands of exceptions that only rarely occur.

© 2018 – Brad A. Myers

"Natural Programming"

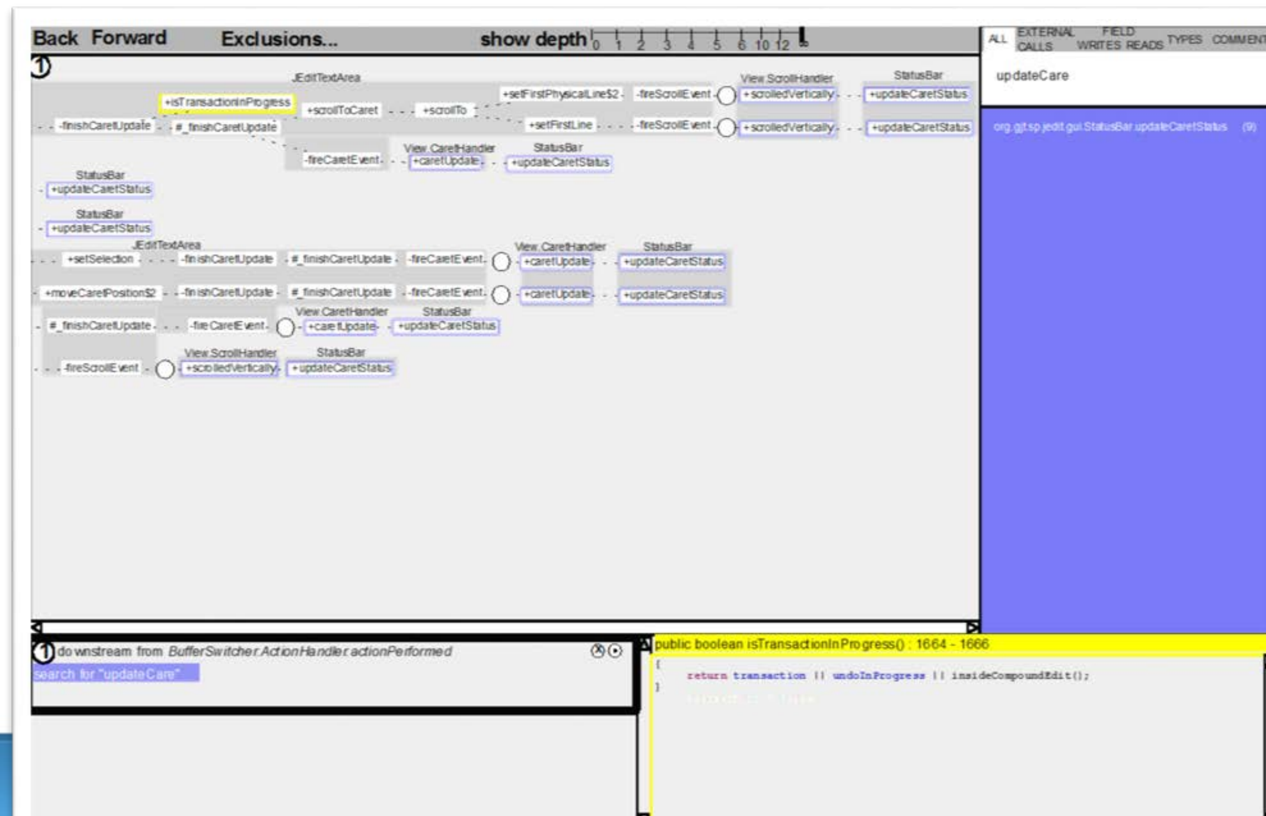
- Technique developed by my group to elicit developer's "natural" expressions
  - Mental models of tasks, vocabulary, etc.
- Blank paper tests
- Must prompt for the tasks in a way that doesn't bias the answers
- Examples:
  - PacMan before and after
    - Mostly rule-based (if-then)
  - API designs
    - Architecture, words used, which methods are on which classes



© 2018 – Brad A. Myers

Early Prototyping

- Thomas LaToza designing new visualization tool to try to help answer Reachability Questions
- Prototypes created with Omnigraffle and printed
- Revealed significant usability problems that were fixed before implementation
  - Graphical presentation
  - Controls



© 2018 – Brad A. Myers

Another Example: Variolite

- How to support data scientists with exploratory programming?
- What kind of version control support would be useful?
  - Interviews and CIs showed that conventional approaches like Git are too heavy-weight
- Showed dozens of sketches to target users to get feedback on which seemed usable and useful
- Resulting design presented at CHI'2017
- **V**ariations **A**ugment **R**eal **I**terative **O**utcomes **L**etting **I**nformation **T**ranscend **E**xploration

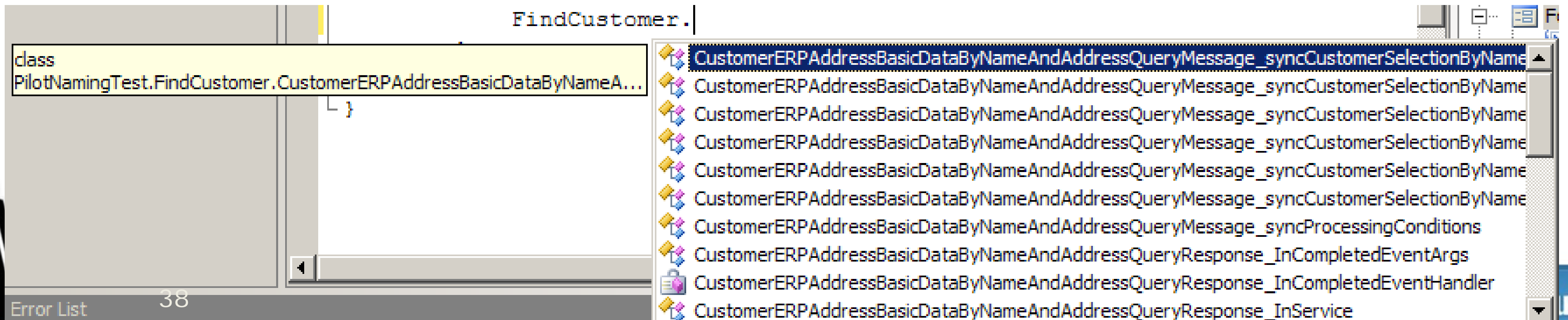
[Kery, Horvath, & Myers, CHI'2017]

```
driverTest.py                                   variants
1  import matplotlib.pyplot as pyplot
2  import numpy as np
3  import math
4
5
6
   Distance1   Distance2   Distance3
7  def distance(x0, y0, x1, y1):
8      return math.sqrt((x1-x0)**2 + (y1-y0)**2)
9
10 def computeAngle (p1, p2):
11     dot = 0
   dot   dot with norm                         variants
12     if computeNorm(p2[0], p2[1]) == 0 or computeNorm(p1[0], p1[1])==0:
13         dot = 0
14     else:
15         dot = (p2[0]*p1[0]+p2[1]*p1[1])
16                 /float(computeNorm(p1[0], p1[1])*computeNorm(p2[0], p2[1]))
17     if dot > 1:
```

© 2018 – Brad A. Myers

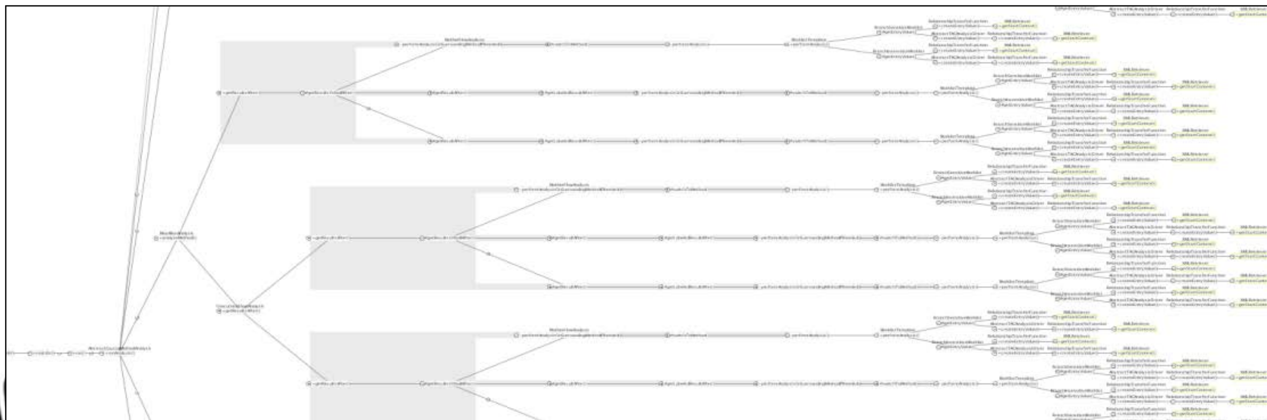
Expert Analyses

- Collaborating with SAP on their APIs and tools
- We studied SAP's Enterprise Service-Oriented Architecture (eSOA) APIs & Documentation
  - Jack Beaton, Sae Young Jeong, Yingyu Xie, Jeffrey Stylos, Brad A. Myers. "Usability Challenges for Enterprise Service-Oriented Architecture APIs," *2008 IEEE Symposium on Visual Languages and Human-Centric Computing, VL/HCC'08*. Sept 15-18, 2008, Herrsching am Ammersee, Germany. pp. 193-196.
- Naming problems:
  - Too long `MaterialSimpleByIDAndDescriptionQueryMessage_syncMaterialSimpleSelectionByIDAndDescriptionSelectionByMaterialDescriptio`
  - Not understandable

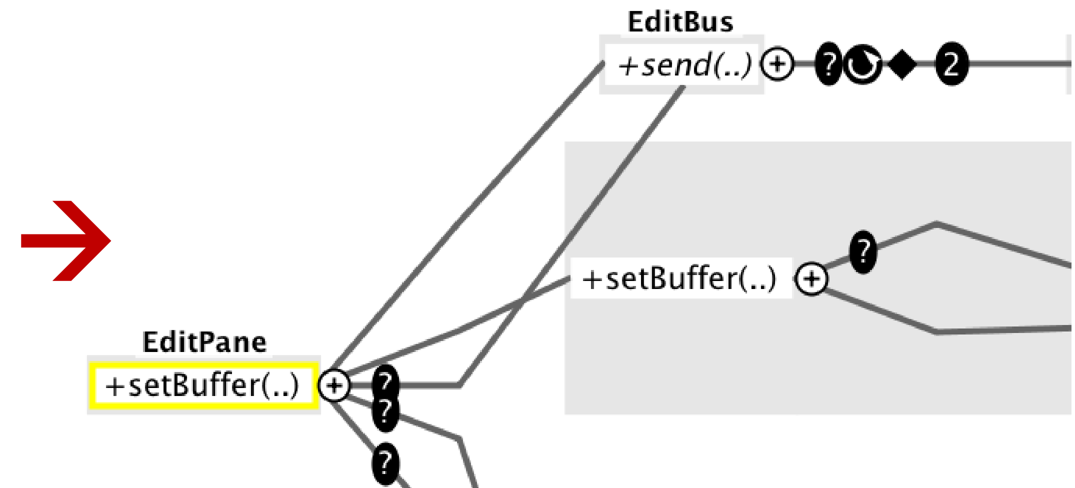


Usability Analysis

- Thomas LaToza's REACHER tool for Reachability Questions went through multiple iterations
  - Revised based on paper prototype (discussed already)
  - Revised based on 1st evaluation of full system
    - E.g., replaced duplicates of calls to methods with pointers
    - Changed to preserve order of outgoing edges
    - Redesign of icons, interactions



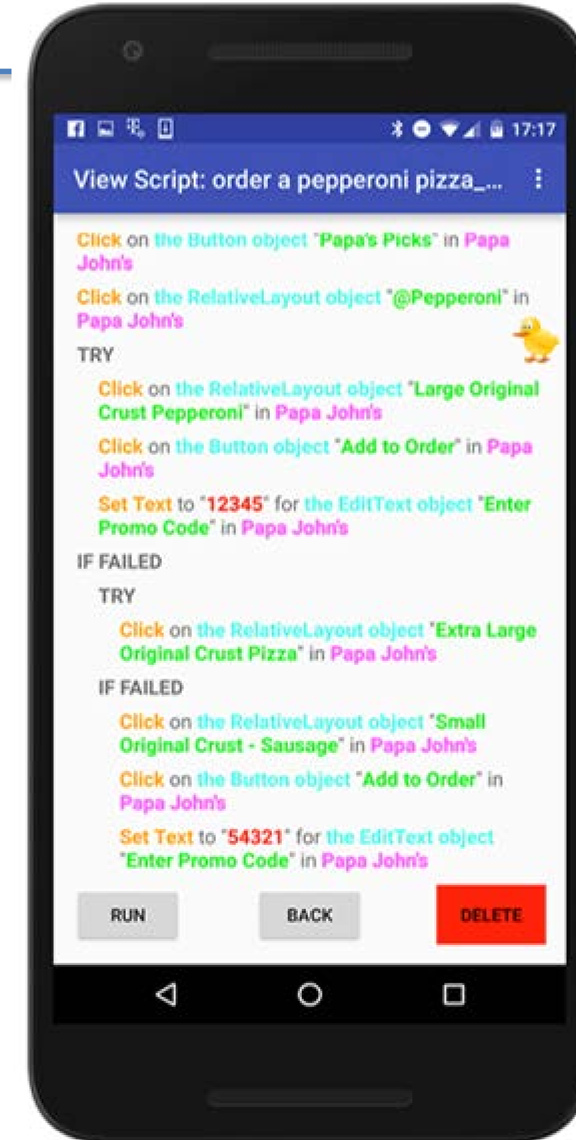
→



© 2018 – Brad A. Myers

Another Example of Usability Analysis

- **Sugilite**: **S**martphone **U**sers **G**enerating **I**ntelligent **L**ikeable **I**nterfaces **T**hrough **E**xamples
- Allow end-users to create automations on Smartphones
- Initiate with speech commands
- Record scripts by example
- Generalizes from one or more examples
- 19 participants attempted 5 tasks
  - All completed at least 2 tasks successfully
  - 8 (42.1%) succeed in all 4 tasks
  - Overall, 65 out of 76 (85.5%) scripts worked
  - Feedback on what we need to improve



A/B Testing of Programming Language Feature

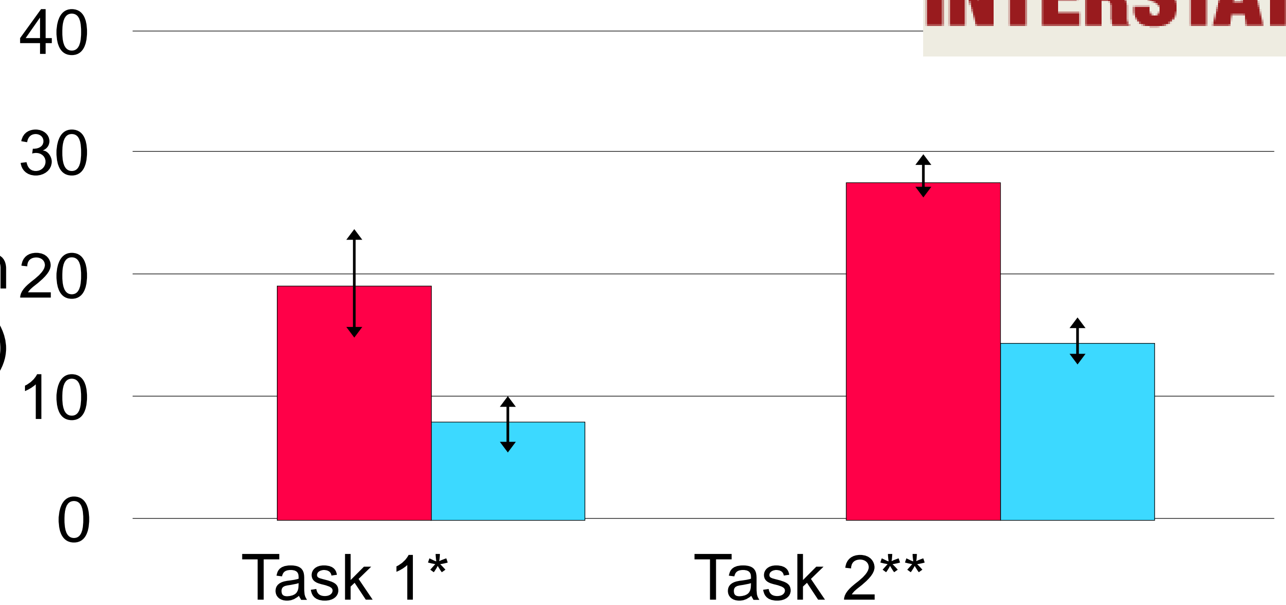
- Glacier immutability extension to Java
- 20 experienced Java programmers
- Compared to using Java `final` as instructed by Josh Bloch

	`final`	Glacier
Users who made errors enforcing immutability (after all tasks)	10/10	0/10
Completed FileRequest.execute() tasks with security vulnerabilities	4/8	0/8
Completed HashBucket.put() tasks with bugs	7/10	0/7

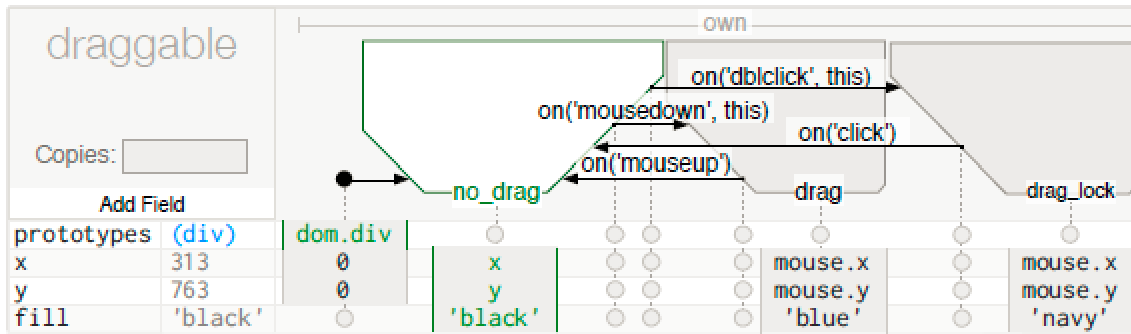
© 2018 – Brad A. Myers

Another Example of A/B testing

- User testing of InterState compared to JavaScript



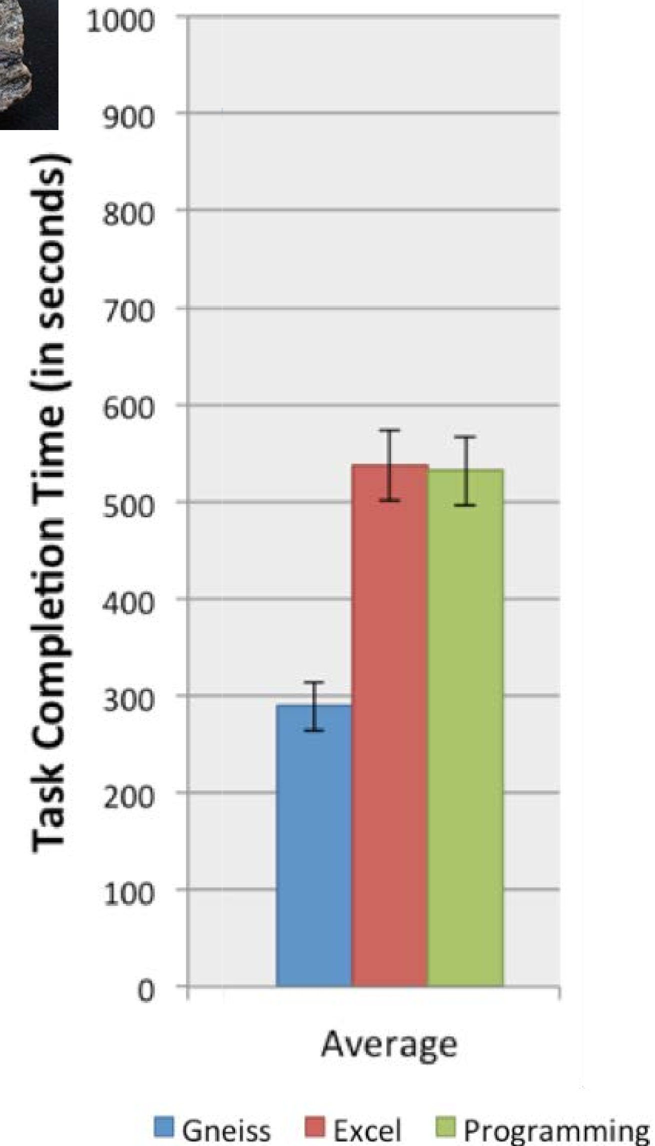

```
var ms_until_advance;
window.setInterval(function ()
    ms_until_advance = 5000 - g
    
    if (diff <= 0) {
        set_selected_index((sel
        reset_timer();
    }
```



© 2018 – Brad A. Myers

Another Example of A/B testing

- Gneiss: **G**athering **N**ovel **E**nd-user **I**nternet **S**ervices using **S**preadsheets
  - Kerry Chang and Brad A. Myers, "Using and Exploring Hierarchical Data in Spreadsheets." *CHI'2016*, pp. 2497-2507.
- Novel spreadsheet interface for investigating hierarchical (e.g., JSON) data
  - Investigate using conventional spreadsheet formulas and drag-and-drop of columns
- Gneiss users significantly outperformed Excel users and programmers ($p<.001$)



1

	A (businesses.name)	B (businesses.categories.cate...)	
1	Coca Cafe	1.1	breakfast_brunch
		1.2	newamerican
2	Waffles Incaffeinated	2.1	breakfast_brunch
		2.2	newamerican
		2.3	tradamerican
3	Point Brugge Café	3.1	belgian
4	The Dor-Stop Restaurant	4.1	breakfast_brunch
		4.2	diners
5	Deluca's Diner	5.1	breakfast_brunch

2

	A (businesses.name)	B (businesses.categories.cate...)	
1	Coca Cafe	1.1	breakfast_brunch
		1.2	newamerican
2	Waffles Incaffeinated	2.1	breakfast_brunch
		2.2	newamerican

column B

3

	A (businesses.categori...)	B (businesses.name)
1	belgian	Point Brugge Café
2	belgian	Park Bruges
3	breakfast_brunch	Coca Cafe
4	breakfast_brunch	Waffles Incaffeinated