Evaluating Programming Languages and Tools in Studies with Human Participants

Thomas LaToza
Motivation

• Evaluate the **usability** of a programming language feature or tool for developers

  • usually productivity effects

• Given a context, what is effect on developer productivity
Challenges

• How many **participants** do I need?
• Is it ok to use students?
• What do I **measure**? How do I measure it?
• What’s an IRB?
• Should I train participants?
• What **tasks** should I pick?
A Practical Guide to Controlled Experiments
Evaluating Software Engineering Tools with Human Participants

Andrew J. Ko, University of Washington, ajko@uw.edu
Thomas D. LaToza, UC Irvine, tlatoza@ics.uci.edu
Margaret M. Burnett, Oregon State University, burnett@eecs.oregonstate.edu

Abstract Empirical studies, often in the form of controlled experiments, have been widely adopted in software engineering research as a way to evaluate the merits of new software engineering tools. However, controlled experiments involving human participants using new tools remain rare. When they are conducted, some have serious validity concerns. Recent research has also shown that many software engineering researchers view this form of tool evaluation as too risky and difficult to conduct, as it might ultimately lead to inconclusive or negative results. In this paper, we aim to help researchers design studies that minimize these risks and increase the quality of controlled experiments with developers by offering practical methodological guidance. We explain, from a practical perspective, options in the recruitment and selection of human participants, informed consent, experimental procedures, demographic measurements, group assignment, training, the selection and design of tasks, the measurement of common outcome variables such as success and time on task, and study debriefing. Throughout, we situate this guidance in the results of a new systematic review of the 345 tool evaluations with human participants that were reported in over 1,700 software engineering papers published from 2001-2011.

Keywords Research methodology, tools, human participants, human subjects, experiments.

1. Introduction

Over the past three decades, empirical studies have become widely accepted as a way to evaluate the strengths and weaknesses of software engineering tools (Zannier et al. 2006, Basili et al. 1986, Basili 1993, Basili 2007, Rombach et al. 1992, Fenton 1993, Tichy et al. 1995, Basili et al. 1999). One of the earliest studies was by Basili et al. (1986). They compared the productivity of software developers using two different tools: a text-based tool and a visual tool. They found that the visual tool led to a significant increase in productivity compared to the text-based tool.
A study of architectural decision practices
Thomas D. LaToza, Ezhelva Shabani, and André van der Hoek
CHASE 2013: Workshop on the Cooperative and Human Aspects of Software Engineering
Reports findings from interviews of developers on their architectural decision practices. Results suggest that architectural decisions are often technology decisions and are sometimes revisited, causing software surprises, following the discovery of an architect's bias.

workshop paper: local pdf, doi, poster

Enabling a classroom design studio with a collaborative sketch design tool
Destyni Lokes, Nicolas Mangano, Thomas LaToza, and André van der Hoek
ICSE 2013: International Conference on Software Engineering, Education Track
Reports findings from a deployment of a distributed interactive sketching tool - Caico - in a software design studio course. Results suggest that Caico enables students to work effectively in teams on design problems and quickly develop, refine, and evaluate designs.

full paper (acceptance rate: 21%) doi

Active code compilation
Cyrus Omar, YoungSack Yoon, Thomas D. LaToza, and Brad A. Myers
ICSE 2012: International Conference on Software Engineering, Education Track
Introduces Aspide, an IDE plugin providing interactive and highly-specialized code generation interfaces through code completion. Reports survey and lab study data on the contexts in which such a system could be useful.

full paper (acceptance rate: 21%) local pdf, doi

How software designers interact with sketches at the whiteboard
Nicolas Mangano, Thomas D. LaToza, Marian Petre, and André van der Hoek
TSE: Transactions on Software Engineering, Feb 2016
Examines how sketches support informal design 'in the moments' through an analysis of the relationships between sketches and the reasoning activities they help to enact.

journal article: local pdf, doi, materials and data

Microtask programming: building software with a crowd
Thomas D. LaToza, W. Ben Towne, Christian M. Adriansen, and André van der Hoek
UIST 2014: Symposium on User interface Software and Technology
Introduces an approach for programming through microtasks and presents CrowdCode, an online IDE for microtask programming.

full paper (acceptance rate: 22%) local pdf, doi

Supporting informal design with interactive whiteboards
Nicolas Mangano, Thomas D. LaToza, Marian Petre, and André van der Hoek
CHI 2014: Conference on Human Factors in Computing Systems
Identifies 14 behaviors to support in informal design at the whiteboard, and reports on three field deployments of an interactive whiteboard conducted to understand the opportunities and challenges in supporting informal design.

full paper (acceptance rate: 25%) local pdf, doi, youtube, github

Questions about object structure during coding activities
Marwan Abi-Antoun, Nahim Ammar, and T. LaToza
CHASE 2010: Workshop on Cooperative and Human Aspects of Software Engineering
Reports data from observations of developers performing maintenance tasks. Identifies typical questions about object structure, examining their frequency and context.

workshop paper: local pdf, doi

Program comprehension as fast finding
Thomas D. LaToza, David Garlan, James D. Herbsleb, and Brad A. Myers
Reports results from observations of complex maintenance tasks. Describes a model of program comprehension and reports findings on the benefits of development experience.

full paper (acceptance rate: 17%) local pdf, doi

Maintaining Mental Models: A Study of Developer Work Habits
Thomas D. LaToza, Gina Veronis, and Robert Delisi
ICSE 2006: International Conference on Software Engineering, Experience Track
Reports results from surveys and interviews of professional software developers. Results reveal developers' use of tools, perceived problems, and practices involving code ownership, rationale, code duplication, and interaction.

full paper (acceptance rate: 18%) local pdf, doi

Understanding and modifying procedural versus object-oriented programs: where does domain knowledge help most?
Thomas D. LaToza and Alex Kitch
Experimentally tests if work with OO code benefits more from domain knowledge than work with procedural code. Results suggest domain knowledge benefits work with procedural code, rather than OO code.

poster: local pdf, conference pdf

A study of architectural decision practices
Thomas D. LaToza, Ezhelva Shabani, and André van der Hoek
CHASE 2013: Workshop on the Cooperative and Human Aspects of Software Engineering
Reports findings from interviews of developers on their architectural decision practices. Results suggest that architectural decisions are often technology decisions and are sometimes revisited, causing software surprises, following the discovery of an architect's bias.

workshop paper: local pdf, doi, poster

Enabling a classroom design studio with a collaborative sketch design tool
Destyni Lokes, Nicolas Mangano, Thomas LaToza, and André van der Hoek
ICSE 2013: International Conference on Software Engineering, Education Track
Reports findings from a deployment of a distributed interactive sketching tool - Caico - in a software design studio course. Results suggest that Caico enables students to work effectively in teams on design problems and quickly develop, refine, and evaluate designs.

full paper (acceptance rate: 21%) doi

Active code compilation
Cyrus Omar, YoungSack Yoon, Thomas D. LaToza, and Brad A. Myers
ICSE 2012: International Conference on Software Engineering, Education Track
Introduces Aspide, an IDE plugin providing interactive and highly-specialized code generation interfaces through code completion. Reports survey and lab study data on the contexts in which such a system could be useful.

full paper (acceptance rate: 21%) local pdf, doi, youtube, project

Visualizing code graphs
Thomas D. LaToza and Brad A. Myers
VL/HCC 2011: Symposium on Visual Languages and Human-Centric Computing
Introduces Reacher, an IDE plugin supporting code investigation. Lab study results suggest Reacher enables developers to answer reachability questions faster and more successfully.

full paper (acceptance rate: 33%) local pdf, doi

Hard-to-answer questions about code
Thomas D. LaToza and Brad A. Myers
PLATEAU 2010: Workshop on the Evaluation and Usability of Programming Languages and Tools
Reports results from a survey of hard-to-answer questions developers ask, revealing 94 questions across 24 categories, many of which are not addressed by existing SE tools.

workshop paper: local pdf, doi

Debunkers ask reachability questions
Thomas D. LaToza and Brad A. Myers
ICSE 2010: International Conference on Software Engineering
Reports results from three studies examining challenges investigating code. Results suggest developers can spend tens of minutes answering a single question, get lost and disoriented, and erroneously make assumptions that result in bugs.

full paper (acceptance rate: 14%) local pdf, doi

Questions about object structure during coding activities
Marwan Abi-Antoun, Nahim Ammar, and T. LaToza
CHASE 2010: Workshop on Cooperative and Human Aspects of Software Engineering
Reports data from observations of developers performing maintenance tasks. Identifies typical questions about object structure, examining their frequency and context.

workshop paper: local pdf, doi

Program comprehension as fast finding
Thomas D. LaToza, David Garlan, James D. Herbsleb, and Brad A. Myers
Reports results from observations of complex maintenance tasks. Describes a model of program comprehension and reports findings on the benefits of development experience.

full paper (acceptance rate: 17%) local pdf, doi

Maintaining Mental Models: A Study of Developer Work Habits
Thomas D. LaToza, Gina Veronis, and Robert Delisi
ICSE 2006: International Conference on Software Engineering, Experience Track
Reports results from surveys and interviews of professional software developers. Results reveal developers' use of tools, perceived problems, and practices involving code ownership, rationale, code duplication, and interaction.

full paper (acceptance rate: 18%) local pdf, doi

Understanding and modifying procedural versus object-oriented programs: where does domain knowledge help most?
Thomas D. LaToza and Alex Kitch
Experimentally tests if work with OO code benefits more from domain knowledge than work with procedural code. Results suggest domain knowledge benefits work with procedural code, rather than OO code.

poster: local pdf, conference pdf
Data on how software engineering community conducts experiments w/ humans

- Systematic review of 1701 software engineering articles
- All papers published at ICSE, FSE, TSE, TOSEM 2001 - 2011

<table>
<thead>
<tr>
<th>Year</th>
<th>described tool</th>
<th>empirical eval</th>
<th>empirical eval w/ humans</th>
</tr>
</thead>
<tbody>
<tr>
<td>2001</td>
<td>1392</td>
<td>63%</td>
<td>17%</td>
</tr>
<tr>
<td>2002</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2003</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2004</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2005</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2006</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2007</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2008</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2009</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2010</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2011</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

% of papers w/ tools w/ empirical evals:

- 2001: 82%
- 2002: 63%
- 2003: 17%

% of human evals by method:

- Experience report
- Lab
- Field
- Interview
- Survey

- 2001: 100%
- 2002: 80%
- 2003: 60%
- 2004: 40%
- 2005: 20%
- 2006: 0%
Controlled experiment

• Only way to argue **causality** - change in var x causes change in var y

• Manipulate **independent** variables
  Creates “conditions” that are being compared
  Can have >1, but # conditions usually exponential in # ind. variables

• Measure dependent variables (a.k.a measures)
  Quantitative variable you calculate from collected data
  E.g., time, # questions, # steps, ...

• **Randomly** assign participants to condition
  Ensure that participants only differ in condition
  Not different in other **confounding** variables

• Test hypotheses
  Change in independent variable causes dependent variable change
  e.g., t-test, ANOVA, other statistical techniques
Anatomy of controlled experiment w/ humans
Terminology

• “Tool” — any intervention manipulating a software developer’s work environment
  
  • e.g., programming language, programming language feature, software development environment feature, build system tool, API design, documentation technique, …

• Data — what you collected in study

• Unit of analysis — individual item of data

• Population — all members that exist

• Construct — some property about member

• Measure — approximation of construct computed from data
Example — Study of shapes

Real world

Study

Population

Sample of population

Constructs

shape
size
filled / empty
color

Measure

is blue?
size >10 or size < 10
(Some) types of validity

- **Validity** = should you believe a result
- **Construct** validity
  - Does measure correspond to construct or something else?
- **External** validity
  - Do results generalize from participants to population?
- **Internal** validity (experiments only)
  - Are the differences between conditions caused only by experimental manipulation and not other variables? (confounds)
Example: Typed vs. untyped languages

S. Hanenberg. (2009). What is the impact of static type systems on programming time? In the PLATEAU workshop, OOPSLA 09.

Participants 26 undergrads Task write a parser 27 hrs

Setup new OO language 16 hr instructions

Conditions type system vs. no type system
found errors at compile time errors detected at runtime

RESULTS

Developers with untyped version significantly faster completing task to same quality level (unit tests).
Example: Study validity

- **Construct** validity
  Does measure correspond to construct or something else?

- **External** validity
  Do results generalize from participants to population?

- **Internal** validity (experiments only)
  Are the differences between conditions caused only by experimental manipulation and not other variables? (confounds)

- **Other** reasons you’re skeptical about results?
Good (not perfect) study designs

• Goals
  Maximize **validity** - often requires more
  more participants, data collected, measures
  longer tasks
  more realistic conditions

• Minimize **cost** - often requires
  fewer participants, data collected, measures
  shorter tasks
  less realistic, easier to replicate conditions

• Studies are **not proofs** - results could always be invalid
  don’t sample all developers / tasks / situations
  measures imperfect

• Goal is to find results that are
  **interesting**
  **relevant** to research questions
  **valid enough** your target audience believes them
Overview

1) recruit

2) test against inclusion criteria

3) consent

4) the experiment procedure

5) gather demographic data

6) assign to group

7) training

8) tasks

   - control condition

   - experimental condition

9) the resulting data set

people who do not participate because they do not fit the inclusion criteria

people who do not participate because they do not consent to participants

id  age  group  time
1  23  control  65
2  27   exp   23
3  29  control  55
4  18   exp   16
5  22  control  43
6  21   exp   13
Deciding who to recruit

- **Inclusion criterion**: attributes participants must have to be included in study

- Goal: reflect characteristics of those that researchers believe would benefit

- Example - Nimmer & Ernst (2002)
  - Support those w/ out experience w/ related analysis tools
  - Chose graduate students
  - Developed items to assess (1) did not have familiarity w/ tool (2) Java experience (3) experience writing code
Common inclusion criteria

• Experience w/ a programming language
  • Self-estimation of expertise; time
• Experience w/ related technologies
  • Important for learning new tool
• Industry experience
  • Indicator of skills & knowledge; could also ask directly
• (Natural) language proficiency
Poor criteria: Paper authors

- **62%** of studies evaluating a tool involved *tool’s authors* using the tool & reporting personal experiences

- Tool designers far more likely to use own tool successfully than those new to tool

- More likely to overlook weaknesses of tool
To use students or not to use students?

- **72%** of 113 SE experiments 1993-2002 used students [Sjoberg 2005]
- 23% reported using students in studies from 2001 - 2011 (many did not report if or if not)
- Students can be too inexperienced to be representative of tools intended users; observer-expectancy effect
- But
  - depends on task & necessary expertise
  - professional masters students may have industry experience
  - can minimize observer-expectancy effect
How many participants to recruit?

- More participants → more statistical power
  - higher chance to observe actual differences
  - power analysis — given assumptions about expected effect size and variation, compute participants number
- Experiments recruited median 36 participants, median 18 per condition
  - Some studies smaller
Recruiting participants

• Marketing problem: how to attract participants meeting inclusion criteria

• Questions:
  • Where do such participants pay attention?
  • What incentives to offer for participation?
Sources of participants

• Students
  • Class announcement, fliers, emailing lists
  • Incentives: small compensation & intrinsic interest

• Software professionals
  • Relationships w/ industry researchers
  • Studies by interns at companies
  • Partnerships or contracts with companies
  • In-house university software teams
  • Meetup developer groups, public mailing lists, FB groups
  • CS Alumni mailing lists, LinkedIn groups
Remote participants

• Online labor markets focused on or including developers (e.g., MTurk, oDesk, TopCoder)

• Pros
  • Can quickly recruit hundreds or thousands of participants
  • Use their own space & tools; work at own time

• Cons
  • May misreport levels of experience
  • Might leave task temporarily; more extraneous variation
Remote participants - MTurk example

• Recruited participants from MTurk across 96 hours

• Used **qualification test** to screen for programming expertise
  
  • multiple choice question about program output

• Paid $5 for <= 30 mins

Participant numbers:

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>4776</td>
<td>3699</td>
<td>999</td>
<td>777</td>
<td>489</td>
<td></td>
</tr>
<tr>
<td>completed</td>
<td>took</td>
<td>qualified</td>
<td>completed 1 task</td>
<td>completed all tasks</td>
<td></td>
</tr>
<tr>
<td>informed consent</td>
<td>qualification test</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Overview
Informed consent

• Enables participants to decide to participate with a few page document

• Key elements

  • Names & contact info for you and other experimenters

  • **Purpose** of the study

  • Brief (one or two sentence) high-level description of the types of work participants will be asked to do

  • Expected **length** of the study

  • A statement of any possible **benefits** or compensation

  • A statement of any possible **risks** or discomforts

  • Overview of the data you will collect (thinkaloud, screencast, survey questions, etc.)

  • Clear statement on **confidentiality** of data (who will have access?)
CONSENT TO ACT AS A HUMAN RESEARCH SUBJECT

Crowd Programming

You are being asked to participate in a research study. Participation is completely voluntary. Please read the information below and ask questions about anything that you do not understand. A researcher listed below will be available to answer your questions.

RESEARCH TEAM
Lead Researcher
Dr. Thomas LaToza
Department of Informatics
tialtoza@uci.edu

Faculty Sponsor
Prof. André van der Hoek
Department of Informatics
andre@ics.uci.edu

Other Researchers
Lee Martie
Christian Adriano
Micky Chen
Luxi Jiang

STUDY LOCATIONS
Your own workspace

STUDY SPONSOR
National Science Foundation

WHY IS THIS RESEARCH STUDY BEING DONE?
The purpose of this research study is to examine how design competitions might be used in crowdsourcing software and user interface design.

HOW MANY PEOPLE WILL TAKE PART IN THIS STUDY?
This study will enroll approximately 40 participants. All study procedures will be conducted in your own workspace.

WHAT PROCEDURES ARE INVOLVED WITH THIS STUDY AND HOW LONG WILL THEY TAKE?
1. You are being asked to participate in a design competition. In the first one-week period, you'll be given instructions for a design task and be asked to submit a design. After submitting your design,
IRB Approval

• US universities have an **Institutional Review Board** (IRB) responsible for ensuring human subjects treated ethically

• Before conducting a human subjects study
  
  • Must complete human subjects **training** (first time only)
  
  • Submit an application to IRB for **approval** (2 - ??? weeks approval time)

• During a study
  
  • Must administer “**informed consent**” describing procedures of study and any risks to participants
Collecting demographic data

- **Goal**: understand expertise, background, tool experience, …

- **Interviews** — potentially more comfortable
  - Before or after tasks

- **Surveys** — more consistent, can be used to test against inclusion criteria during recruiting
Assigning participants to an experimental condition

- Random assignment
  - distributes random variation in participant skills and behavior across all conditions
  - minimizes chance that observed difference is due to participant differences
- Used with a between-subjects experiment
- Are alternative designs that can reduce number of participants necessary to recruit
Within-subjects design

- All participants use all tools being compared one at a time across several tasks
  - e.g., participant uses tool in task 1 but not task 2

- **Learning effect** — doing first task may increase performance on second task

- **Counterbalancing** — randomize order of task & on which task participants use each tool

- Latin Square design
Interrupted time-series design

- Measure outcome variable **before** tool introduced, **after** introduced, **after removing** tool
- Can see possible causal effects of tool
- Enables participants to articulate effects of tool
- Could be “trial run” of new tool in a field deployment of tool to a company
Training participants

- Knowledge participants need includes:
  - how to use **tools** in the environment provided
  - terminology & domain **knowledge** used in task
  - design of programs they will work with during task

- Can provide background and **tutorial** materials to ensure participants have knowledge.
To train or not to train?

• Key study design question, creating assumptions about context of use results generalize to

• Training
  • Ensures participants are proficient and focused on the task

• No training
  • Generalizes directly to new users who don’t have training materials, but risks study being dominated by learning

• Studies often choose to provide training materials for tool
Design of training materials

• Goal: teach required concepts quickly & effectively

• Possible approaches
  
  • Background materials
  
  • Video instructions
  
  • Tutorial where participants complete example task w/ tool
  
  • Cheat sheets

• Can also include assessment to ensure learning

• Can be helpful for experimenter to answer participant questions
Overview

1) recruit

2) test against inclusion criteria

people who do not participate because they do not fit the inclusion criteria

3) consent

4) the experiment procedure

5) gather demographic data

6) assign to group

7) training

8) tasks

control condition

experimental condition

9) the resulting data set

id | age | group | time
---|-----|-------|------
1  | 23  | control | 65
2  | 27  | exp    | 23
3  | 29  | control | 55
4  | 18  | exp    | 16
5  | 22  | control | 43
6  | 21  | exp    | 13
Tasks

• Goal: design tasks that have coverage of work affected by tool

• Key tradeoff: realism vs. control
  • How are real, messy programming tasks distilled into brief, accessible, actionable activities?
  • More realism —> messier, fewer controls
  • More control —> cleaner, less realism

• Tradeoff often takes the form of tradeoff between bigger tasks vs. smaller tasks
Feature coverage

- Of all functionality and features of tool, which will receive focus in tasks?

- More features $\rightarrow$ more to learn, more variation in performance, higher risk of undue negative results

- Fewer features $\rightarrow$ less to learn, less ecological validity, more likely to observe differences
Experimental setting

• Experiments can be conducted in lab or in developer’s actual workspace

• Experiments most often conducted in lab (86%)
  • Enables control over environment
  • Can minimize distractions
  • But less realism, as may have different computer, software, … from participants’ normal setting
Task origin

• **Found** task — task from real project (15%)
  - e.g., bug fix task from an OSS project
  - More *ecologically* valid
  - May not exist for new tools
  - Can be hard to determine what feature usage found task will lead to

• **Synthetic** task — designed task (85%)
  - Can be easier to tailor for effective feature *coverage*
  - Must compare synthetic task to real tasks
Task duration

• **Unlimited** time to work on a task
  
  • Allow either participant or experimenter to determine when task is complete
  
  • Hard to find participants willing to work for longer time periods

• **Fixed** time limit
  
  • More control over how participants allocate time across tasks
  
  • Can introduce floor effect in time measures, where no one can complete task in time

• Typical length of **1 - 2** hours
Measuring outcomes

- Wide range of possible measures
  - Task completion, time on task, mistakes
  - Failure detection, search effort
  - Accuracy, precision, correctness, quality
  - Program comprehension, confidence
- Most frequent: success on task, time on task, tool usefulness
Measuring success on task

• Often multiple ways to succeed
  • e.g., several ways to implement feature or fix bug
• What is close enough to be counted as success?
• Might be binary success measure
• Might be measure of quality of change
Determining when goal is reached

- Experimenter *watches* participant for success
  - Requires *consistency*, which can be challenging
- Success is *automatically* measured (e.g., unit tests)
  - Requires researcher to identify all goal states in advance, which can be challenging
- **Participants** determine they believe they have succeeded
  - Most ecologically valid
  - Introduces variation, as participants may vary in confidence they obtain before reporting they are done
Defining success to participants

- Need to **unambiguously** communicate goal to participants

- When participants themselves determine, may ask experimenter about what is success
  - Experimenter can reiterate **instructions** from beginning

- When experimenter determines
  - Experimenter should respond “I unable to answer that question”
Measuring time on task

• Need to define task **start** and task **end** & who determines when task has finished
  
  • Choice of task framing

• What is **start**
  
  • When participant starts reading task — includes variation in time spent reading
  
  • When participants starts working

• What is **end**
  
  • What happens if participant succeeds but does not realize it?
  
  • What happens if they think they succeeded but failed?
Measuring usefulness

• Usefulness — does the tool provide functionality that satisfies a user need or provides a benefit
  • Not usability — ease of use for task

• Might ask developers
  • Did they find the tool useful
  • Would they consider using it in the future

• Technology Acceptance Model
  • Validated instrument for measuring usefulness through a questionnaire
Debriefing & compensation

• Explain to participant what study investigated

• Explain the correct solutions to tasks

• Instructions about information that should not be shared with others
  • e.g., don’t share tasks with friends who might participate

• Get speculative feedback about tool
  • Can use semi-structured interview to get perceptions of tool
Piloting

• Most **important** step in ensuring useful results!

• (1) Run study on **small** (1 - 4) number of participants

• (2) Fix **problems** with study design
  Was the tool tutorial sufficient?
  Did tasks use your tool? Enough?
  Did they understand your materials?
  Did you collect the right data?
  Are your measures correct?

(3) Fix **usability** problems
  Are developers doing the “real” task, or messing with tool?
  Are users confused by terminology in tool?
  Do supported commands match commands users expect?

• (4) **Repeat** 1, 2, and 3 until no more (serious) problems
Overview

1) recruit

2) test against inclusion criteria

3) consent

4) the experiment procedure

5) gather demographic data

6) assign to group

7) training

8) tasks

   control condition

   experimental condition

9) the resulting data set

10) debrief

people who do not participate because they do not fit the inclusion criteria

people who do not participate because they do not consent to participants

id  age  group  time
1   23   control  65
2   27   exp     23
3   29   control 55
4   18   exp     16
5   22   control 43
6   21   exp     13

50
Qualitative data
On the value of qualitative data

• Experiment may provide evidence that A is “better” than B

• But always generalizability questions about why and when

• Qualitative data offers possibility of explanation, making it possible to explain why result occurred.

• Can use coding to convert qualitative data to categorical data, which can be counted or associated with time to create quantitative data
Collecting qualitative data

• **Screencasts**
  • **Record** screen as participants do tasks
  • Many video recorders (e.g., SnagIt)
  • Offers insight into **what** participants did
    • What was time consuming
  • Permits quantitative analysis of **steps & actions**
    • Can code more fine-grained time data
  • Does not provide insight into why developers did what they did
Collecting qualitative data

• Think-aloud
  • Ask participants to **verbalize** what they are thinking as they work
  • **Prompt** participants when they stop talking for more than a minute or two
  • Offers insight into **why** participants are doing what they are doing
    • What barriers are preventing progress on task
Analyzing qualitative data

1. **open** coding - read through the text
   look for **interesting** things relevant to research questions
   add notes in the margin (or column of spreadsheet)
   add “**codes**” naming what you saw
   make up codes as you go, not systematic

2. **axial** coding - how are codes related to each other?
   look for **patterns**: causality, ordering, alternatives

3. **selective** coding - from initial codes, select interesting ones
   which codes found interesting things?
   from initial examples, build definition on when they are applied
   **systematically** reanalyze data and apply codes

4. **second** coder (optional)
   2nd person independently applies codes from definitions
   check for interrater **reliability** - if low, iterate defns & try again
Example
REACHER: Interactive, compact visualization of control flow
Evaluation

Does REACHER enable developers to answer reachability questions faster or more successfully?

Method

12 developers  
15 minutes to answer reachability question × 6

Eclipse only on 3 tasks  
Eclipse w/ REACHER on 3 tasks  
(order counterbalanced)

Tasks

Based on developer questions in prior observations of developers.

Example:

When a new view is created in jEdit.newView(View), what messages, in what order, may be sent on the EditBus (EditBus.send())?
Developers with REACHER were 5.6 times more successful than those working with Eclipse only.

(not enough successful to compare time)

Task time includes only participants that succeeded.
REACHER helped developers stay oriented

Participants with REACHER used it to jump between methods.

“It seems pretty cool if you can navigate your way around a complex graph.”

When not using REACHER, participants often reported being lost and confused.

“Where am I? I’m so lost.”

“These call stacks are horrible.”

“There was a call to it here somewhere, but I don’t remember the path.”

“I’m just too lost.”

Participants reported that they liked working with REACHER.

“I like it a lot. It seems like an easy way to navigate the code. And the view maps to more of how I think of the call hierarchy.”

“Reacher was my hero. ... It’s a lot more fun to use and look at.”

“You don’t have to think as much.”
Conclusions

• Controlled experiments w/ humans can demonstrate *causal* relationship between tool & productivity effects of tool

• But… observed in *context* where study conducted

• Key role for more research to understand *representativeness* of context

• High value in qualitative understanding of productivity effects to help bridge this gulf
Resources


