Design Process

SWE 795, Fall 2019
Software Engineering Environments
Today

• Part 1 (Lecture)(~1 hr)
  • Design process: what are the steps in a developer-centered approach to designing developer tools?
    • Exploratory study: what challenges does a developer face?
    • Sketching & prototyping: how might a new tool better address these challenges?
    • Experiments: does your tool help developers work more effectively than they were before?
  
• Part 2 (In-Class Activity)(~30 mins)
  • Conducting an observational exploratory study
• Break!
• Part 3 (Discussion)(30 mins)
  • Discussion of WhyLine and Programmers are Users Too papers (will discuss 3rd paper during lecture section)
• Questions about your project
  • Ask me after class, drop by office hours Wed 3-4:30, or make appt
Logistics

- 2.5 weeks: HW1 — 9/23 — lit review + revised tool idea
- 2 weeks: HW2 — 10/7 — observational study
- 2 weeks: HW3 — 10/21 — sketches of tool interaction
- 6 weeks: HW4 - 12/2 — tool + small (2-3 participants) study

- No readings on days HW assignments due.
  - Will use time for HW presentations
- Presentation of readings on
  - 9/16 — 2 readings -- **SIGNUP TODAY**
  - 9/30 — 3 readings
  - 10/28 — 3 readings
  - 11/4 — 3 readings
  - 11/11 — 3 readings
  - 11/18 — 3 readings
  - 11/25 — 3 readings
Reading presentation grading rubric

• 5 min presentation, lead 10 min discussion with class

• Effectively identifies the key insight, approach, and results of the paper

• Clear and effective communication style

• Effective time management, limiting summary to 5 minutes

• Effectively stimulates discussion of paper with class
Exploratory studies

- **Field observations / ethnography / lab observations**
  - Observe developers at work
- **Surveys**
  - Ask *many* developers specific questions
- **Interviews**
  - Ask a *few* developers *open-ended* questions
- **Contextual inquiry**
  - Ask *questions* while developers do work
- **Indirect observations (artifact studies)**
  - Study artifacts (e.g., code, code history, bugs, emails, ...)

- **Observe** developers at work
- **Many** developers specific questions
- **Few** developers *open-ended* questions
- **Questions** while developers do work
Exploratory studies: goals

• Understand the process that developers use to tackle a programming problem
  • What questions do developers ask?
  • What strategies do they use to answer these questions?
• Identify steps that are time consuming
• Identify barriers that prevent developers from making progress
• Identify breakdowns, where developers’ mental model diverges from system (e.g., inserting defects)
• In what ways do tools support or not support?
Controlled experiment

- Only way to argue **causality** - change in var x causes change in var y
  - Often used to test impact of a tool
  - Does change in programming tool cause change in {time, success, ...}

- 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
Study design
Anatomy of a user study
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

Population

Study

Sample of population

Constructs

Measure

shape
size
filled / empty
color

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 vs. 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
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
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
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?)
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
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
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 —> more to learn, more variation in performance, higher risk of undue negative results

- Fewer features —> 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
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
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
Conducting an observational user study
Introduction

• Greet participants, introduce yourself, thank them
• Build rapport, socialize
• Introduce them to the setup
• Relieve anxiety and curiosity as much as possible
• Make clear evaluating design, not participant
• Let participants know you can’t answer questions about how to do task
Starting session

- Give participants description of task
- Start any video recording
- Start encouraging participant to think aloud
- Begin observing participants work on task
Interactions during the task

• Goal: listen, not talk
• Prompt participants to think aloud when necessary
  • e.g., What are you trying to do? What did you expect to happen?
• If show signs of stress / fatigue, let them take a break
• Keep participants at ease
  • If participants frustrated, reassure & calm participants
  • If so frustrated they want to quit, let them
Giving help

• If participants totally off track, small reminder of goal might help
• Should **not** give participants information about how to complete the task
• What if user asks for help?
  • Direct them to think through it or work it out for themselves
Collecting critical incidents

• Any action that does not lead to progress in performing the desired task
• May sometimes be related to a gulf of execution or gulf of evaluation
• Generally does not include
  • accessing help
  • random acts of curiosity or exploration
  • slips
Understanding a critical incident

• Important to understand in the moment what users goal is and what actions they are taking
• When a critical incident occurs, jot down
  • The time
  • What user was trying to do
  • What user did
Wrapping up the study session

• Provide questionnaire (if applicable) / conduct interview (if applicable)
• Answer any lingering questions the participant may have
• Thank the participant!!
• Provide any incentives (if applicable)
Reset study environment

• Make sure study environment is in the same state for all participants
  • Reset browser history / cache (if applicable)
  • Delete any user created content or materials
Transcript: example

SwitchWindow     updateCaretStatus()
ReferencesTo     View.CaretHandler.caretUpdate()
5:34
Edit             comments out updateCaretStatus() call
Edit             comments out guards

ErrorTo          View.getViewConfig()
BackTo           updateCaretStatus()
SwitchWindow     View.getViewConfig()
5:35
BackTo           View.CaretHandler.caretUpdate()
Edit             put guards back, adds logging statement

ReferencesTo     View.handleEditPanelUpdate()
Edit             comment out call to updateCaretStatus()
ReferencesTo     View.setEditPanel()
Edit             comment out call to updateCaretStatus()
ReferencesTo     StatusBar.handleMessage()
5:36
Edit             comment out call to updateCaretStatus()
SwitchWindow     View.setEditPanel()
Run/Edit
Hit breakpoint   updateCaretStatus()
Sketching & Prototyping
Sketches are Sketchy

• Not mechanically correct and perfectly straight lines

• **Freehand**, open gestures

• Strokes may miss connections

• Resolution & detail **low** enough to suggest is concept

• Deliberately **ambiguous** & abstract, leaving “holes” for imagination

- Annotations explain what is going on in each part of sketch & how
Sketches support design exploration

- Physical interactions
  - Mouse, keyboard, touchscreen
  - Physical software interactions
    - What things are on screen
    - What things are active

- Navigation
  - Right/left click
  - Backward, forward
  - Opening, closing
  - Saving, undoing

- Learning the basics
  - Titlebar, toolbar, taskbar
  - Menubar

- Ways to teach them stuff
  - Learn by example
  - How do users get confident
  - How do you ask someone
    - "Is this your first time using a PC?"
    - Without being annoying?
    - What about OMs or OMs overriding everything?

- If you need to know one thing it’s this...
  - What am I cool?

- Show me

- Problem 1: figuring out the expertise of someone
  - Problem 2: knowing what they need help with
  - Problem 3: building a UI that they can do it
Fidelity of sketches & mockups

storyboard

wireframe

prototype

low

(Many details left unspecified)

fidelity

high

(more polished & detailed)
Storyboards for UI design

- Sequence of visual “frames” illustrating **interplay** between user & envisioned system
- Explains how app fits into a larger **context** through a single scenario / story
- Bring design to **life** in graphical clips - freeze frame sketches of user interactions
- “Comic-book” style **illustration** of a scenario, with actors, screens, interaction, & dialog
Crafting a storyboard

• Set the stage:
  • Who? What Where? Why? When?
• Show key interactions with application
• Show consequences of taking actions
• May also think about errors
Example: ticket kiosk

Ticket buyer walks up to the kiosk

Displays “Occupied” sign on wraparound case

Sensor detects user & starts immersive process

Detects people with ID card
Example: ticket kiosk

Greets buyer and asks for PIN

Buyer selects “Boston symphony at Burruss Hall”

Shows recommendations & most popular categories

Plays music from symphony, shows date & time picker
Frame transitions

• Transitions between frames particularly important
• What users think, how users choose actions
• Many problems can occur here (e.g., gulfs of execution & evaluation)
• Useful to think about how these work, can add thought bubbles to describe
In Class Activity
Think-Aloud Usability Study

• In groups of 2
  • Conduct a small think-aloud usability study.
  • One person will serve as participant. Other as observer.
  • Observer will ask participant to complete a short programming task while engaged in think-aloud.
    • If participants forgets to think-aloud, prompt them, e.g., "What are you working on now?"
  • Observer will take notes on activity, notes key steps and any critical incidents that occur.