Usability Studies

Michael Coblenz

Today

- Think-aloud usability studies: a way to gather ground truth about challenges people face when using your software

- Additional resource: https://www.nngroup.com/articles/usability-testing-101/

A TOUR OF QUALITATIVE METHODS

- Data sources

  - Interviews and focus groups

  - **Usability studies**

  - Surveys

  - Contextual inquiry

  - Corpus studies

- Analytic approaches

  - Thematic analysis

  - (others)

INTERVIEWS AND FOCUS GROUPS

- Method: make a list of questions. Ask them 1-1 or to a group.

- Useful when you want to learn from experts

- Results depend on interview skill and quality of participants

USABILITY STUDIES

- Method: ask participants to do tasks with a system. Observe what problems they have.

- RQ: "What challenges do users have when they do X?"

- Great for iterating on designs

- Depends on availability of suitable users and tasks

SURVEYS

- Useful for gathering data from many people

- Not great for depth

CONTEXTUAL INQUIRY

- Watch someone doing a task

- Depends on finding an expert

CORPUS STUDIES

- RQ: "How often does X occur in the wild?"

  - or: "Does X ever occur in the wild?"

- e.g., X = null pointer dereference bugs

- e.g., X = harassment of open-source contributors

- Requires an X detector (maybe manual analysis) and a corpus

ANALYSIS

- Many qualitative studies produce textual data

  - Interview transcripts

  - Bug reports

  - Code snippets

  - Images

- Can we do better than "I read it and it seems to me…"?

IN PRACTICE

- Industrial user studies are usually informal.

- Not trying to produce generalizable results or convince others.

- But I'm going to hint at a more structured way anyway.

OPEN-CODE THE DATA

- Meaning: categorize each element

- Manual process

- Can parallelize (have multiple coders)

  - Then have to worry about consistency

- Now you have categories!

THEMATIC ANALYSIS

- (danger: this summary is incomplete)

- In brief: repeatedly group codes until you have a hierarchy

- Top-level groups are "themes"

Running Studies

STUDY DESIGN OVERVIEW

- Running studies requires:

  - If *research* study: ethics approval (but this is not a research class)

  - Recruiting

  - Training

  - Task design

  - Data collection/analysis

ETHICS REVIEW

- For research: need to submit proposal to Institutional Review Board (IRB)

- For this class: no need to get IRB approval (IRB only supervises *research*)

https://about.citiprogram.org/

ETHICS

- What if incentive is too high?

- What if incentive is too low?

  - IRB reviews incentives

- What if recruitment is misleading?

  - IRB reviews recruitment materials

PARTICIPANT PRE-SCREENING

- Can issue a pre-test to avoid wasting time on unqualified participants.

- How will you incentivize people to take the test?

Which of the following might be a valid Java constructor invocation?

malloc(sizeof(Square))

Square.new(5)

square(5)

new Square(5)

In Java, *encapsulation* refers to:

Preventing clients from improperly depending on

Serializing data correctly so that it is transmitted

Using the `capsule` keyword to protect secret da

```
void test() {
  ArrayList list1 = new ArrayList(
  list1.add(1);

  ArrayList list2 = list1;
  list2.add(2);

  System.out.println(list1.size())
}
```

If `test()` is run, what is the output?

1

2

Do not use any external resources to answer this question.

Which statements are true of interfaces in standard Java?

	True	False
Interfaces have no field declarations unless they are `public static final`.	○	○
Methods in interfaces are public by default.	○	○
Methods in interfaces (except for `default` methods) lack bodies.	○	○
A class can implement no more than one interface.	○	○

DEMOGRAPHICS

- Collect information if you want it!

- Programming experience? Languages?

- If they tell you, you can use it…

- e.g. Gender_____

TRAINING

- How will you prepare your participants?

- People don't read.

- People think they understand but in fact do not.

- Teach…and then assess.

- Or: decide that no training is necessary.

Search docs

Docs » Obsidian Tutorial

 Edit on GitHub

Obsidian Tutorial

- Ownership – Introduction
  - Principles of ownership
- Ownership – Transactions
  - Transaction return types
  - Transaction parameters
  - Transaction receivers (`this`)
- Ownership – Variables
  - Assignment
  - Fields
  - Local variables
  - Constructors
- Ownership – Miscellaneous
  - Ownership checks
  - Getting rid of ownership
  - Invoking transactions
  - Handling Errors
  - Return
- Assets
- States – Introduction
  - States and Ownership
- States – Manipulating State
  - The `->` Operator
  - Alternative field initialization
  - Optional compiler checks
  - Testing states with `in`
- States – Miscellaneous
  - Unowned references
  - Shared references
  - Implicit casts
- States and Assets
- Using Obsidian on a Blockchain
  - Concurrency

Write a contract called **Person** that has an **Owned** reference to a **House** and a **Shared** reference to a **Park**. The **House** and **Park** contracts are given below.

```
contract House {


}

contract Park {


}
```

Please write your answer in the VSCode window (code1.obs). You may compile your code to check your answer.

```
contract Money {
    ...
}

contract Wallet {
    Money@Owned m;

    Wallet@Owned() {
        m = new Money();
    }

    transaction spendMoney() returns Money@Owned {
        ...
    }

    transaction receiveMoney(Money@Owned >> Unowned mon) {
        ...
    }
}
```

What is **m** in the above code fragment above?

○ A Money object

○ An Owned reference to a Money object

○ An Owned object

○ All of the above

○ None of the above

RECRUITMENT

* Flyers

* Emails

* Social network

* Buy ads

* The street

See: Report from Dagstuhl Seminar 19231
Empirical Evaluation of Secure Development
Processes

INCENTIVES

- $$$ (in person, MTurk)

- Desire to contribute to science / help you out

- Food

- Fame (leaderboard)

- Rare experience

- Learning opportunity

- Distraction from work

- Credit

THINK-ALOUD USABILITY STUDIES

- Give people tasks and observe what happens.

- NOT experiments

- NOT comparative

- Just want to see what problems people encounter.

- Follow "think-aloud" protocol

USABILITY STUDIES CAN SHOW

- Participants encountered the following problems…

- Participants were confused by…

- Only participants who knew X were able to do the task.

USABILITY STUDIES CANNOT SHOW

- My system is better than an existing system.

USABILITY STUDY TASKS

- Choose an *interesting* task

  - One that you think might be hard

  - One that is central to the usability of your design

- Can't test everything

WHY TASKS?

- Opinions are often not convincing

- Hypothetical questions are especially unconvincing

- Need to see what *actually happens* when users do *realistic tasks*

TASKS

- This is the hardest part of study design.

- You will not get this right the first time.

- Solution: pilot repeatedly.

- What is the distribution over task times?

TASK IDEAS

- Write a program according to this specification.

- Are there bugs in this code? If so, what are they?

- Fill in the missing code…

- What does this code do?

- Answer these questions about this code.

TASK DESIGN

- Must carefully restrict tasks!

- People will get stuck on irrelevant things

- Decide how much help to provide

- Ideally: scope task to focus on the variable of interest

- *Constrain the task as much as possible.*

DATA COLLECTION

- Think-aloud

- Audio recordings

- Videos

- Screen capture

- Eye tracking

- Post-study survey

•Take lots of notes!, including timestamps! You do not want to watch the videos.
•Include a clock on the screen.

THINK-ALOUD

- Two varieties: concurrent and retrospective

- "Please keep talking."

- Can't use timing as a dependent variable due to effect of explanations.

TASK CONTEXTS

- Pencil/paper

- Text editor

- IDE

- Compiler?

- Debugger?

- Test cases?

YOUR TURN

- You are interested in studying challenges that users have when using the Gmail web app.

- Design tasks that you will give your participants in a 30-minute study.

CONCLUSION

- Running usability studies requires:

  - Recruiting

  - Training

  - Task design

  - Data collection/analysis

- Task design is probably the trickiest. Start early and pilot!