

FEEDBACK

- Way too long. Cut the measure usability slides in the middle.

Talk to people!

17-313: Foundations of Software **Engineering**

<https://cmu-313.github.io>

Michael Hilton and Josh Sunshine

Fall 2026

Gentle Reminder

- Please close laptops
- Tablets with stylus ok



Administrivia

- **P3 - Interviews & Requirements.** Due Thursday, October 29th, 11:59pm
- Schedule your three interviews **today**
 - One Course staff: sign up on the Slack sheet (to be released after class)
 - One non-team member
 - One non-class member
- Midsemester Grades Posted (attendance misses prorated)
- Midterm 1 regrade window is open
- Problem set due tomorrow (as a push to the reci repo)
- Quiz next Tuesday

Learning Goals

- Identify examples of bad design and explain, from the user's perspective, why they fail
- Choose a user study method that fits the question you are asking
- Plan and run an interview that produces checkable expectations, not opinions
- Turn interview evidence into requirements that are testable and traceable

What is usability?

*“the extent to which a product can be used by specified users to achieve specified goals with **effectiveness, efficiency and satisfaction** in a specified context of use.”*

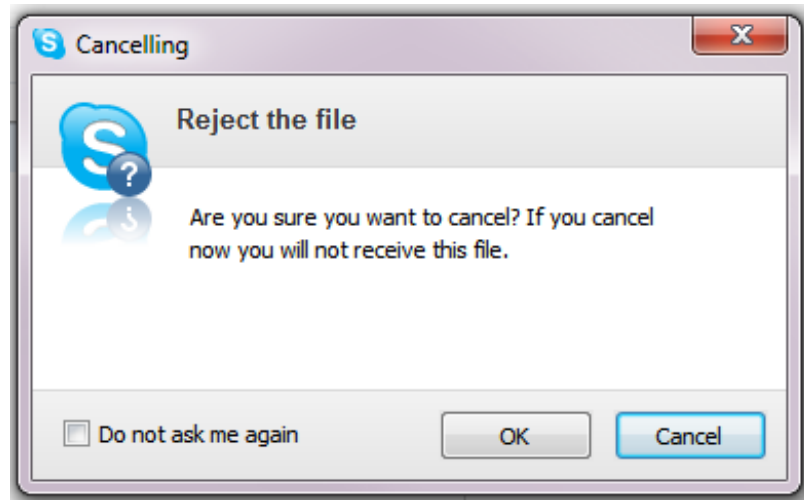
ISO 9241-11

Learnability, Efficiency, Memorability, Errors, Satisfaction





- New York overhauled their signs
- Left is before, right is after
- Is it better? Is it good?





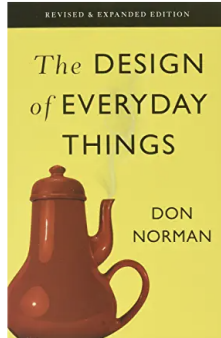




Same handle! Different action?



Best Seller



The Design of Everyday Things: Revised and Expanded Edition

by [Don Norman](#) | Nov 5, 2013

★★★★☆ ~ 1,434

Paperback

\$14⁴⁹

✓**prime** FREE Delivery **Sat, Sep 12**

More Buying Choices

\$9.47 ([59 used & new offers](#))

Audible Audiobook

\$0⁰⁰ ~~\$28.50~~

Free with Audible trial

Kindle

\$11⁹⁹ ~~\$18.99~~

Plus, earn a \$3.00 credit.

Available instantly

Great On Kindle: A high quality digital reading

#1 book on this topic

**Ok, innocent enough. But
can anything bad happen?**



EMERGENCY ALERTS

now

Emergency Alert

BALLISTIC MISSILE THREAT INBOUND TO HAWAII. SEEK IMMEDIATE SHELTER. THIS IS NOT A DRILL.

Slide for more

BMD False Alarm

Amber Alert (CAE) - Kauai County Only

Amber Alert (CAE) Statewide

1. TEST Message

PACOM (CDW) - STATE ONLY

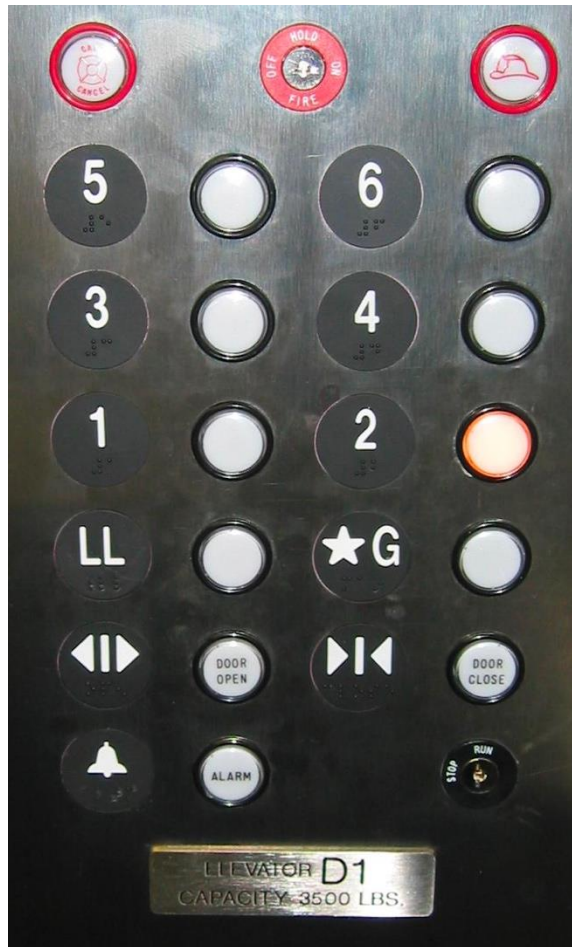
Tsunami Warning (CEM) - STATE ONLY

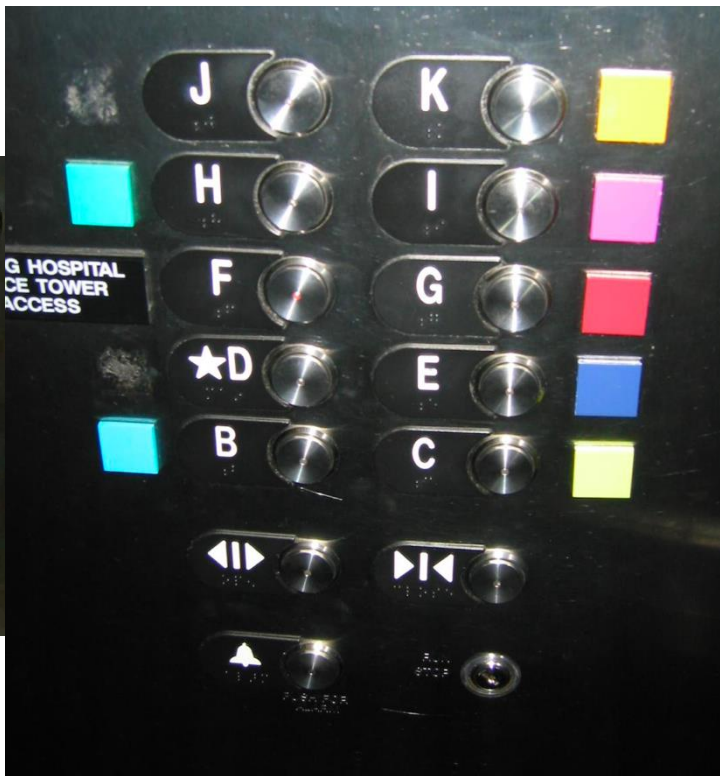
DRILL - PACOM (CDW) - STATE ONLY

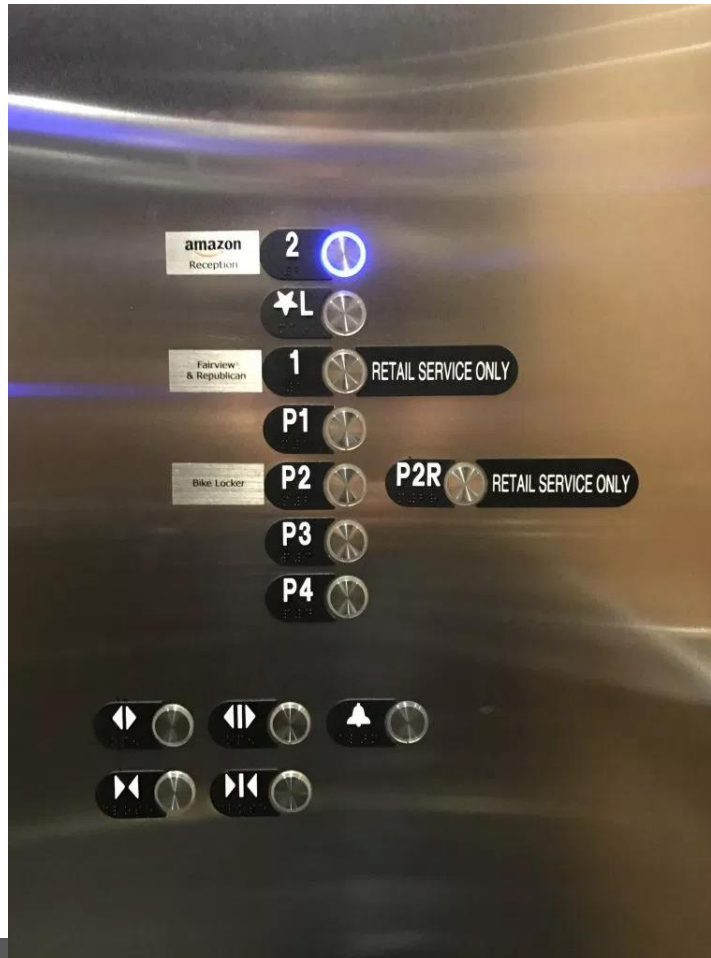


Therac-25

Ever look at elevator button panels?





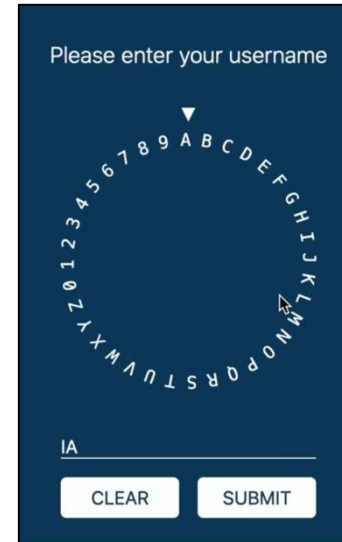


Did you read the instructions?



Design for people





r/BadUIBattles

For more funny examples of bad design...

Not just Graphics!

```
+ 1311c40...d5b9653 gh-pages -> origin/gh-pages (forced update)
hint: You have divergent branches and need to specify how to reconcile them.
hint: You can do so by running one of the following commands sometime before
hint: your next pull:
hint:
hint:   git config pull.rebase false  # merge
hint:   git config pull.rebase true   # rebase
hint:   git config pull.ff only       # fast-forward only
hint:
hint: You can replace "git config" with "git config --global" to set a default
hint: preference for all repositories. You can also pass --rebase, --no-rebase,
hint: or --ff-only on the command line to override the configured default per
hint: invocation.
fatal: Need to specify how to reconcile divergent branches.
michaelhilton@MICHAELHMACM1 cmu-17-356.github.io % git config pull.rebase false
michaelhilton@MICHAELHMACM1 cmu-17-356.github.io % git pull
fatal: refusing to merge unrelated histories
```



COVER FEATURE **21ST CENTURY USER INTERFACES**

Programmers Are Users Too: Human-Centered Methods for Improving Programming Tools

Brad A. Myers, Carnegie Mellon University

Amy J. Ko, University of Washington

Thomas D. LaToza, George Mason University

YoungSeok Yoon, Google

PLIERS: A Process that Integrates User-Centered Methods into Programming Language Design

MICHAEL COBLENZ, Carnegie Mellon University

GAURI KAMBHATLA, University of Michigan

PAULETTE KORONKEVICH, University of British Columbia

JENNA L. WISE, Carnegie Mellon University

CELESTE BARNABY, Facebook, Inc.

JOSHUA SUNSHINE, JONATHAN ALDRICH, and BRAD A. MYERS,

Carnegie Mellon University

Programming language design requires making many usability-related design decisions. However, existing HCI methods can be impractical to apply to programming languages: languages have high iteration costs, programmers require significant learning time, and user performance has high variance. To address these problems, we adapted both formative and summative HCI methods to make them more suitable for programming language design. We integrated these methods into a new process, PLIERS, for designing programming languages in a user-centered way. We assessed PLIERS by using it to design two new programming languages. Glacier extends Java to enable programmers to express immutability properties effectively and easily. Obsidian is a language for blockchains that includes verification of critical safety properties. Empirical studies showed that the PLIERS process resulted in languages that could be used effectively by many programmers and revealed additional opportunities for language improvement.

CCS Concepts: • **Human-centered computing** → *User studies; Usability testing; User centered design*; • **Software and its engineering** → *Object oriented languages; Data types and structures*;

What usability problems have you seen with software?

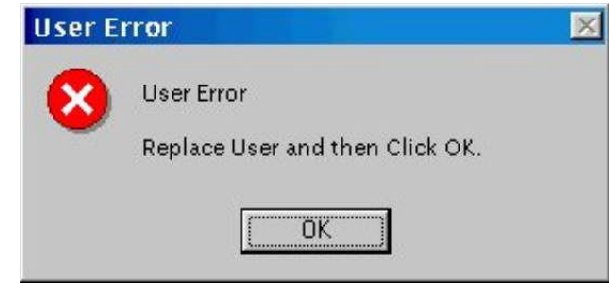
How do we decide what to build next?

How do we decide what to build next?

- Intuition?
- Test on ourselves?
- Copy interface from other apps?
- React to problems?

Test with actual USERS

- Put your app in front of actual people
- Can they figure it out?
- How long does it take them?
- Do they need help?



- **How they behave != how you want them to behave**
 - Don't blame the user!
 - Blame the design and go fix it!

How do you measure usability?

- Imagine your group project...
- How do you measure whether it is usable?
- How do you test it with people?

Hallway user testing

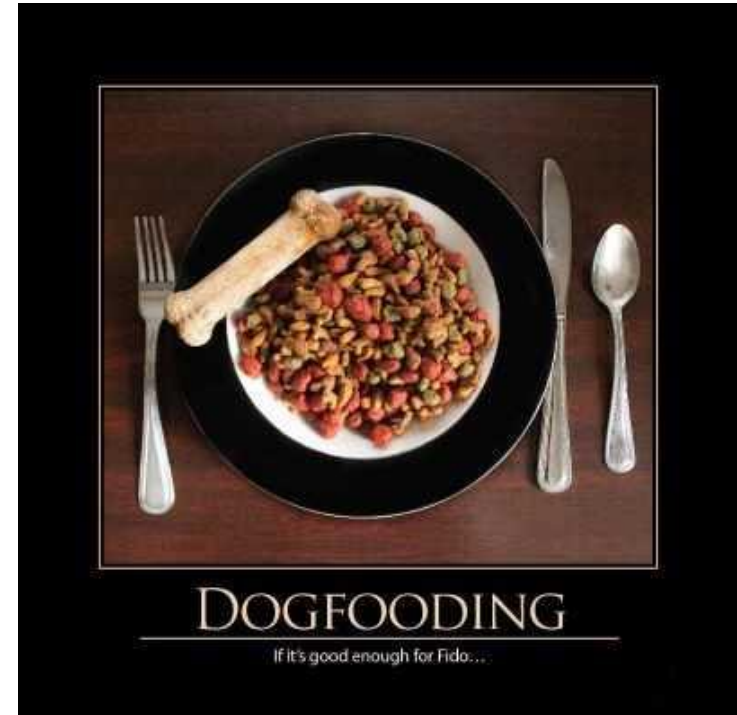
- Ask the next person who walks down the hall.
- Can they figure it out without help?
- Give them a task with it... do they do it as you expected?



What problems might there be with this method?

“Dogfooding”

- Use your own product.
- Helps close the gap between user and developer.



User study methods

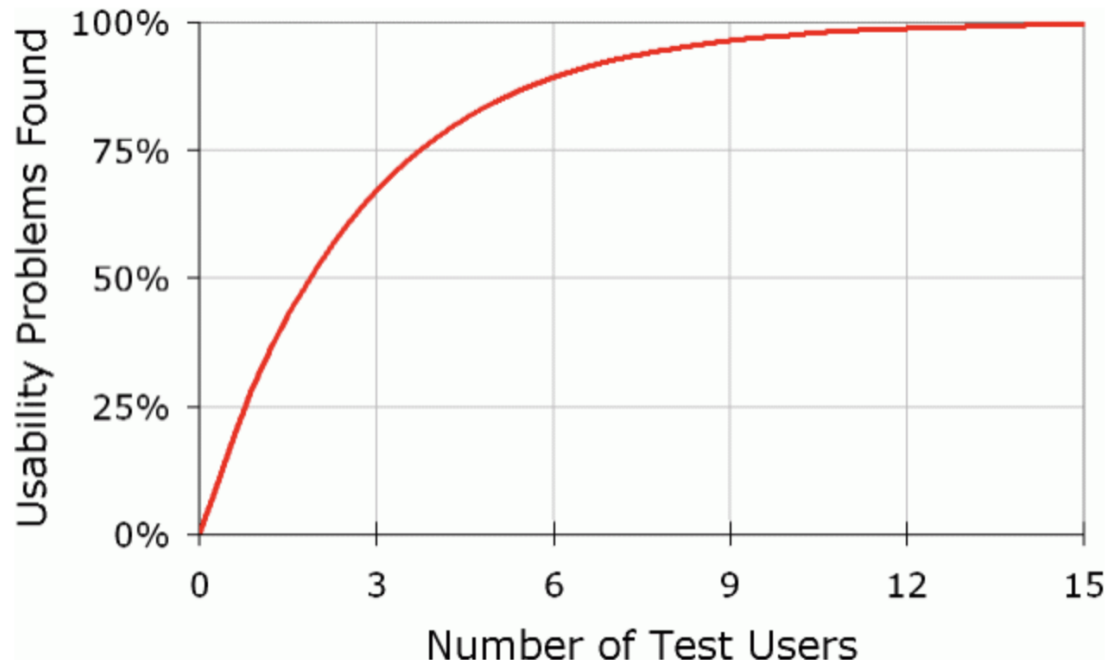
- Interviews
- Surveys
- Shadowing
- Lab study
- Field study
- Wizard of Oz
- Focus group
- Retrospective study
- Diary study
- A/B testing
- Pros/cons of each
- Each answers different questions
- Often use multiple

Lab study

- Controlled setting with complete control over tasks
- Common to record the participant and the screen
- Can be formal or informal
- Many possibly tasks
 - Based on your user stories
 - Ask them to complete the same task using two different apps
- Metrics: task time, correctness, mistakes, opinions
- Beware of reactivity effects
- Do they use the app as you expected?



Revealing usability concerns



A/B testing



A/B Testing

- Compare two variations of a design
- Often used for web pages with a call to action
- Randomly show a user design A or B
- Does design A or B lead to more sales/signups/clicks?
- Requires many users
- Is there a *statistically significant difference*?
- Doesn't tell you *why*

Shadowing

- “Fly on the wall”
- Observe people in their own setting... often work
- Rich qualitative data
- Expensive!
- Time consuming
- Difficult to get access



Interview & Survey

- Open-ended questions show the users perspective
- Users will surprise you
- Don't ask leading questions
 - *"How much do you love my app?"*
- Responses don't always match behavior
- Scalable
- *"Do you want this feature?" "YES!"*



Prototyping

- Implementing a full feature/product is expensive
- Prototype to gather requirements first
- Mockups/wireframes
- Paper prototype
- Wizard of Oz

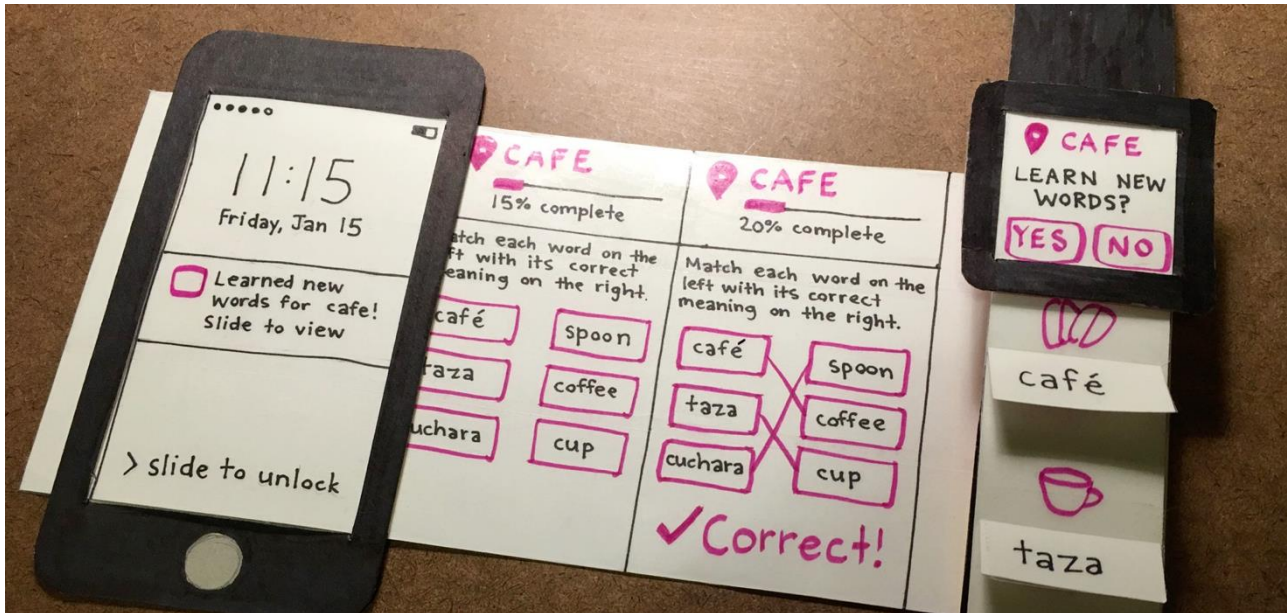
Prototyping

- Implementing a full feature/product is ~~expensive~~ Easy with AI!
- Prototype to gather requirements first
- Mockups/wireframes
- Paper prototype
- Wizard of Oz

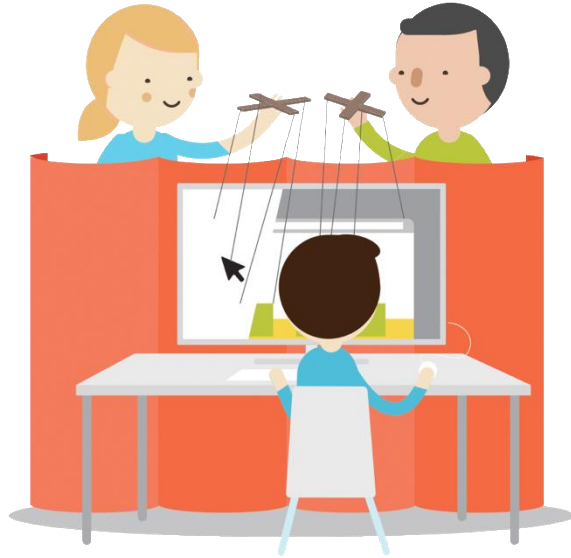
No longer true!!



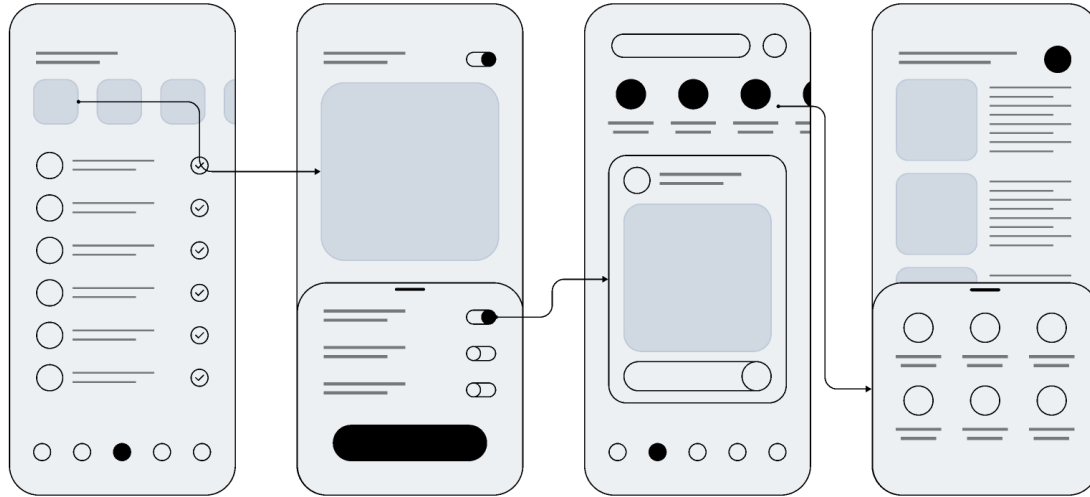
Paper prototype



Wizard of Oz



Wireframes



Aspects of Usability

Learnability



Efficiency



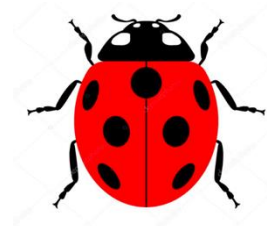
Satisfaction



Memorability

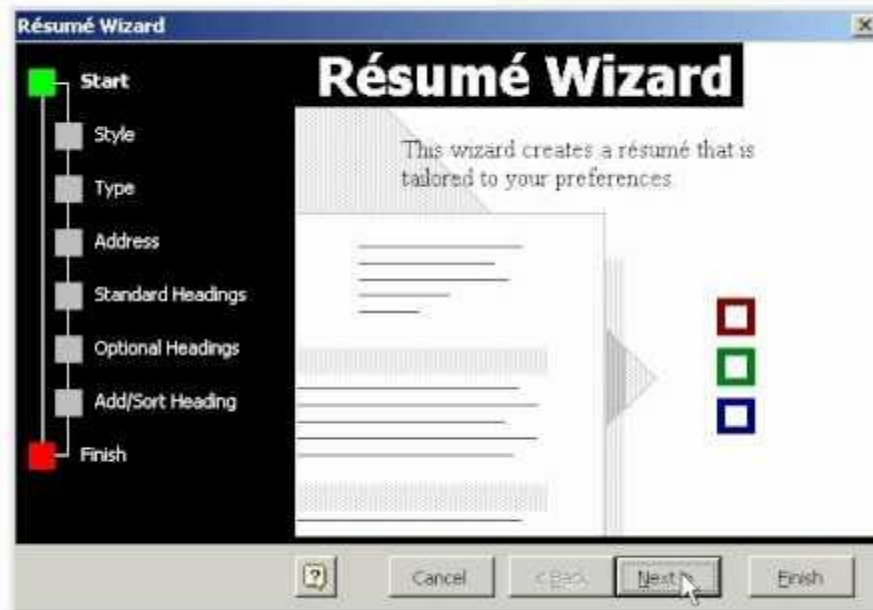


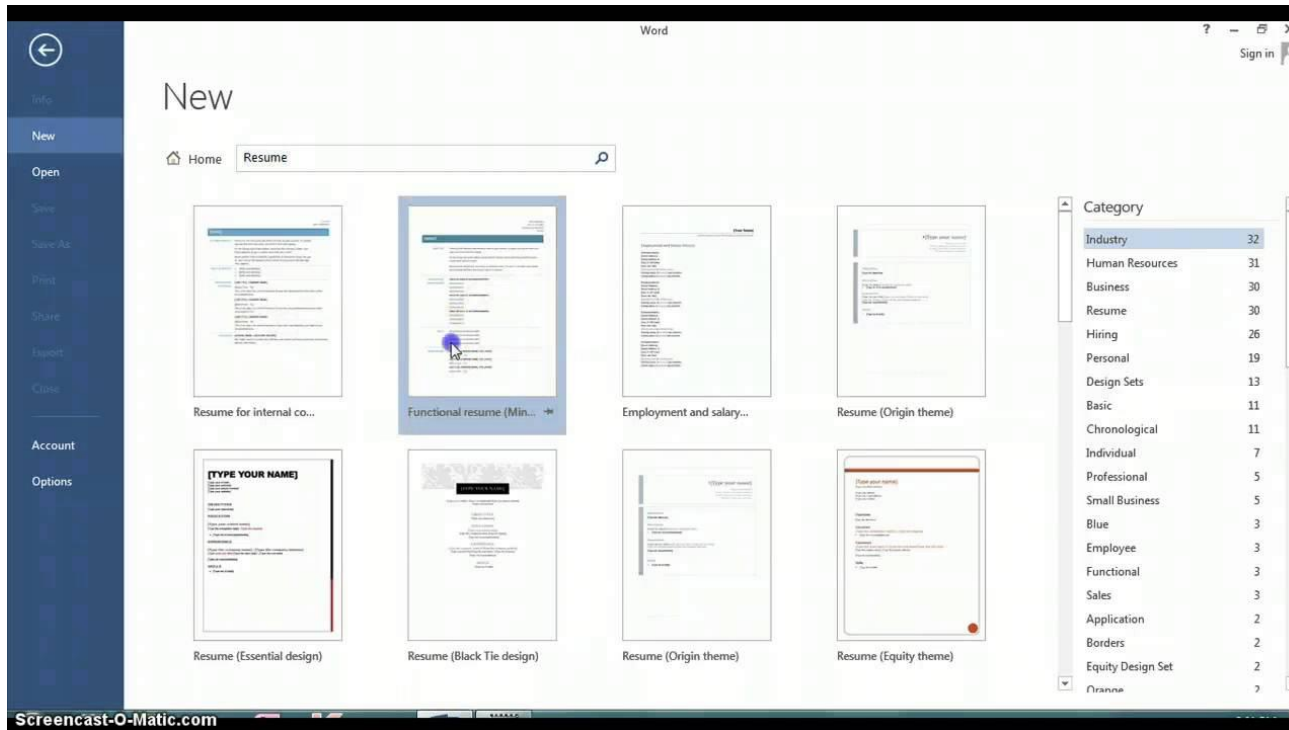
Errors



Learnability

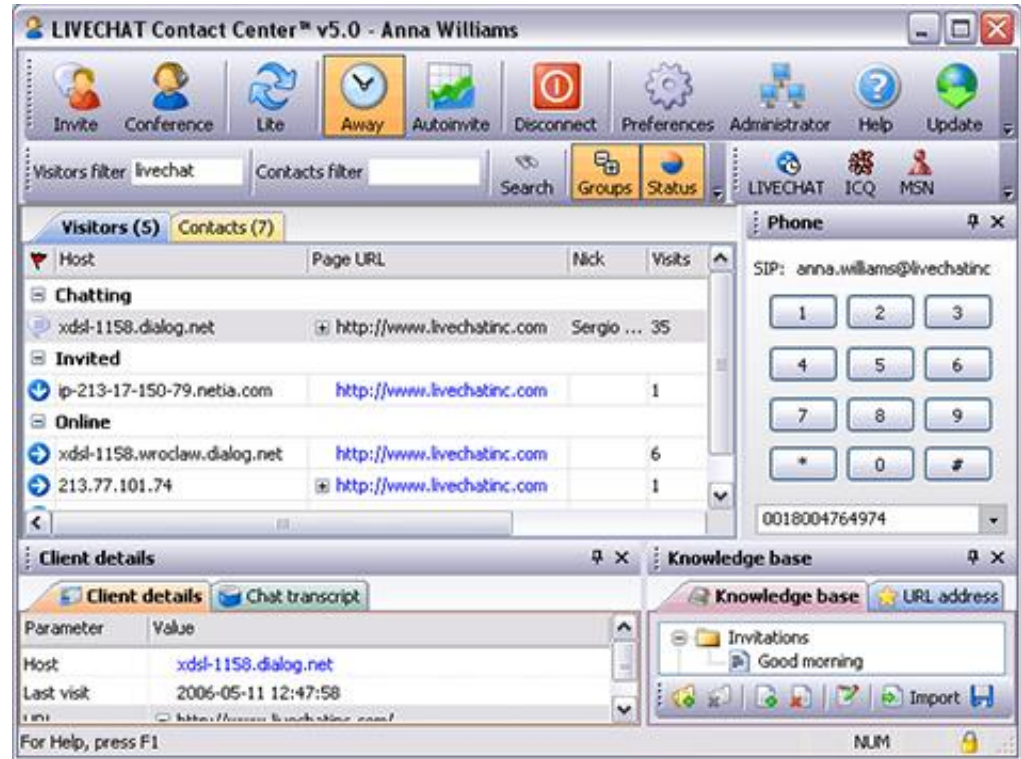
- Can users learn to correctly use it?
 - Can unfamiliar users pick it up?
 - Can power users learn to become experts with it?
- How to provide the right guidance to help users?
- They won't read instruction books!
- Wizards?
- Visible & labeled buttons





Design constraint:

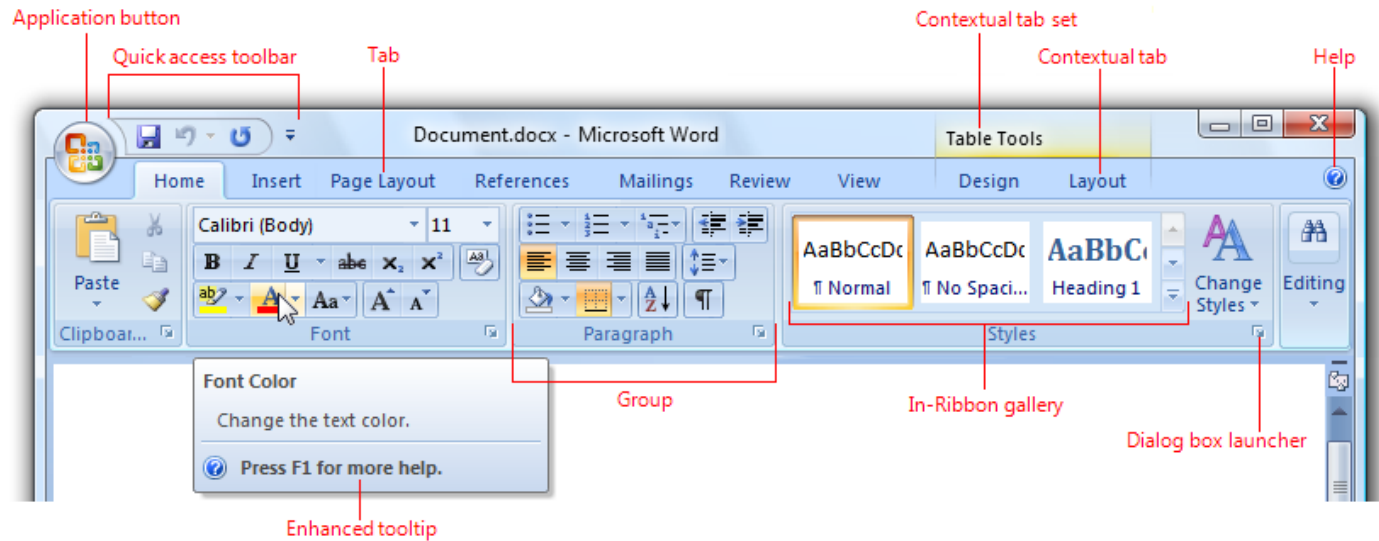
- Don't overwhelm the user
- Make it easy to find features



How could we measure learnability?

Efficiency

- How fast can users perform the correct interaction?
- How do you design interfaces for faster interactions?



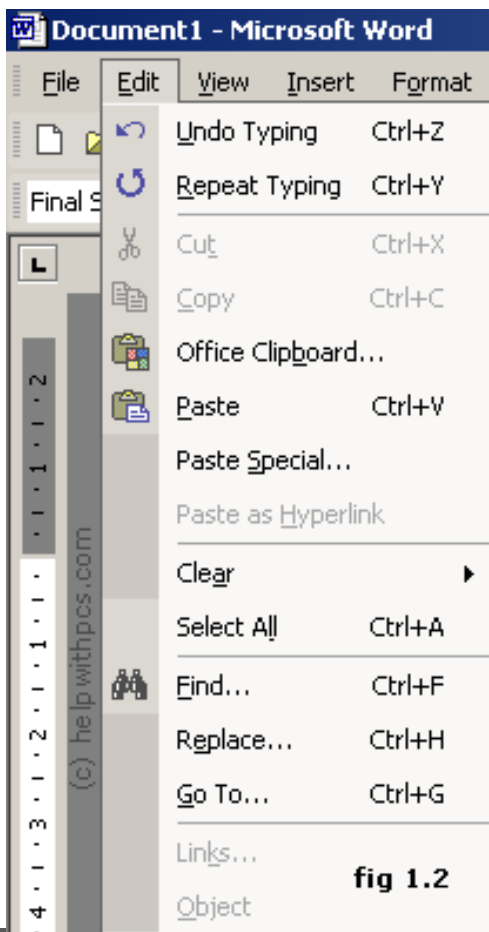
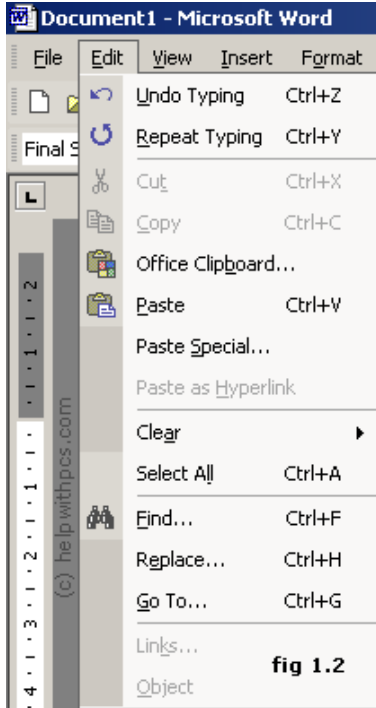


fig 1.2

```
write — -zsh — 80x24
drwxr-xr-x  59 azh  staff  1.8K Jan 21 12:56 src
drwxr-xr-x  53 azh  staff  1.7K Jan 21 12:56 test
drwxr-xr-x   3 azh  staff   96B Jan 21 12:56 types
-rw-r--r--   1 azh  staff  1.9K Jan 21 12:56 webpack.common.js
-rw-r--r--   1 azh  staff  195B Jan 21 12:56 webpack.dev.js
-rw-r--r--   1 azh  staff  397B Jan 21 12:56 webpack.installer.js
-rw-r--r--   1 azh  staff  472B Jan 21 12:56 webpack.prod.js
[~/Coding/NodeBB] λ cd src/routes/write
[~/Coding/NodeBB/src/routes/write] λ ll
total 120
-rw-r--r--   1 azh  staff  1.5K Jan 21 12:56 admin.js
-rw-r--r--   1 azh  staff  2.2K Jan 21 12:56 categories.js
-rw-r--r--   1 azh  staff  4.4K Jan 21 12:56 chats.js
-rw-r--r--   1 azh  staff  993B Jan 21 12:56 files.js
-rw-r--r--   1 azh  staff  1.2K Jan 21 12:56 flags.js
-rw-r--r--   1 azh  staff  2.4K Jan 21 12:56 groups.js
-rw-r--r--   1 azh  staff  2.4K Jan 21 12:56 index.js
-rw-r--r--   1 azh  staff  2.8K Jan 21 12:56 posts.js
-rw-r--r--   1 azh  staff  967B Jan 21 12:56 search.js
-rw-r--r--   1 azh  staff  542B Jan 21 12:56 tags.js
-rw-r--r--   1 azh  staff  3.8K Jan 21 12:56 topics.js
-rw-r--r--   1 azh  staff  4.7K Jan 21 12:56 users.js
-rw-r--r--   1 azh  staff  557B Jan 21 12:56 utilities.js
[~/Coding/NodeBB/src/routes/write] λ
```

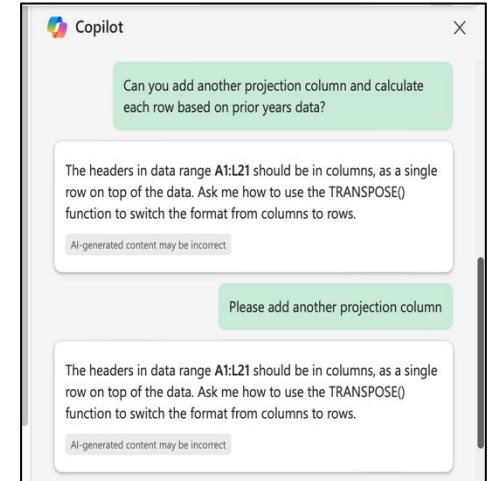
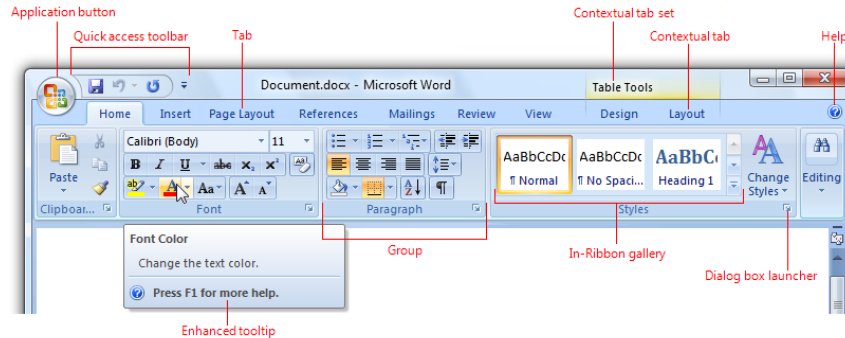
```

write --zsh-- 80x24
drwxr-xr-x 59 azh staff 1.8K Jan 21 12:56 src
drwxr-xr-x 53 azh staff 1.7K Jan 21 12:56 test
drwxr-xr-x 3 azh staff 96B Jan 21 12:56 types
-rw-r--r-- 1 azh staff 1.9K Jan 21 12:56 webpack.common.js
-rw-r--r-- 1 azh staff 195B Jan 21 12:56 webpack.dev.js
-rw-r--r-- 1 azh staff 397B Jan 21 12:56 webpack.installer.js
-rw-r--r-- 1 azh staff 472B Jan 21 12:56 webpack.prod.js

[~/Coding/NodeBB] λ cd src/routes/write
[~/Coding/NodeBB/src/routes/write] λ ll
total 120
-rw-r--r-- 1 azh staff 1.5K Jan 21 12:56 admin.js
-rw-r--r-- 1 azh staff 2.2K Jan 21 12:56 categories.js
-rw-r--r-- 1 azh staff 4.4K Jan 21 12:56 chats.js
-rw-r--r-- 1 azh staff 993B Jan 21 12:56 files.js
-rw-r--r-- 1 azh staff 1.2K Jan 21 12:56 flags.js
-rw-r--r-- 1 azh staff 2.4K Jan 21 12:56 groups.js
-rw-r--r-- 1 azh staff 2.4K Jan 21 12:56 index.js
-rw-r--r-- 1 azh staff 2.8K Jan 21 12:56 posts.js
-rw-r--r-- 1 azh staff 967B Jan 21 12:56 search.js
-rw-r--r-- 1 azh staff 542B Jan 21 12:56 tags.js
-rw-r--r-- 1 azh staff 3.8K Jan 21 12:56 topics.js
-rw-r--r-- 1 azh staff 4.7K Jan 21 12:56 users.js
-rw-r--r-- 1 azh staff 557B Jan 21 12:56 utilities.js

[~/Coding/NodeBB/src/routes/write] λ

```



Design constraint:

- Let experts be fast
- Let novices learn



How could we measure efficiency?

Errors

- Is it unlikely that users will make errors?
- Is it easy to recover from errors?



Make it hard to make mistakes

- Unambiguous buttons
- Clear descriptions
- No surprises!
- Every action should be revertible*

Revertible!

- Ever close a tab on accident?
 - Reopen tab
- Ever close all your tabs?
 - Reopen all
- Ever close Chrome?
 - Recent update made it harder
- Delete a file?
 - Trash
- Changed a file?
 - Undo
- Clicked the Next button?
 - Previous



For permanent things...
Ask if they are sure.
Consider a delay.

How could we measure errors?

Interviews: from opinions to requirements

P3: Elicit Requirements

- Pick one opencode feature
- Write an interview script
- Interview three different kinds of users
- Turn what you learn into testable requirements
- **Due Thursday, October 29th, 11:59pm**

**“Users like it” is not a
requirement**

What would an engineer check?

- *"It should keep my secrets out."*
 - A file matching a `.gitignore` pattern never shows up as untracked in `git status`
- *"It should be easy to debug."*
 - `git check-ignore -v <file>` names the rule and line that ignored it
- *"It should work in subfolders."*
 - A pattern with no slash, like `*.log`, matches files at any depth

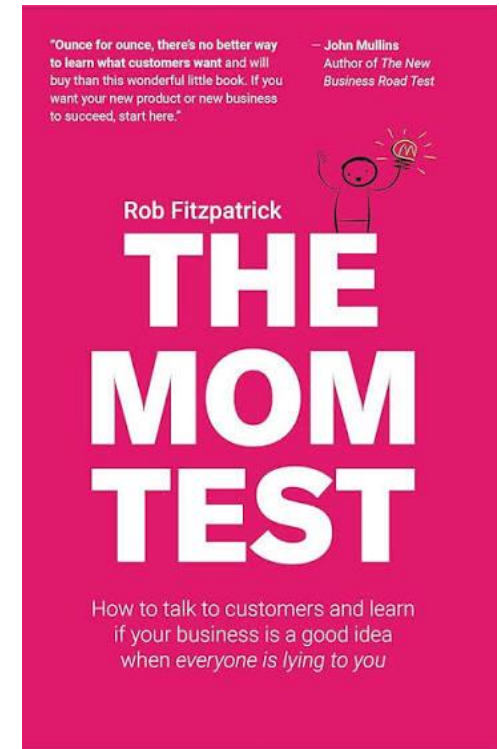
What makes a requirement useful?

- **Testable:** someone can check it and say “holds” or “doesn’t hold”
- **Traceable:** points to the interview lines it came from
- **Situated:** names the input, state, or action
- **About behavior:** what the user sees, not how to build it
- **Acceptance criteria** = a concrete situation + the expected outcome

How do you get people to tell you the truth?

The Mom Test

- Rob Fitzpatrick, 2013
- Written for startup founders talking to customers
- Problem: **people are nice to you**
 - Ask if your idea is good, they'll say yes
- Ask questions that **even your mom couldn't lie to you about**



Bad: pitching your idea

You: “Mom, I’m building a cookbook app for the iPad. Would you buy it?”

Mom: “Oh, that sounds wonderful! How much?”

Better: asking about her life

You: “What did you last use your iPad for?”

Mom: “Email, mostly. And the crossword.”

You: “When did you last try a new recipe? Where did you find it?”

Mom: “Last week. A magazine. I never look online.”

Three rules

- **Talk about their life**, not your idea
- **Ask about specifics in the past**, not opinions about the future
 - “When did that last happen?” beats “Would you ever...?”
- **Talk less**, listen more

But P3 requires a demo!

- Rule 1 says don't pitch your feature
- So start with how they work today
 - Which AI coding tools? What do they let the tool do on its own?
 - When did a tool last do something they didn't want?
- **Then** demo the feature

Three kinds of bad data

(1) Compliments

- Sounds like: *“Oh, that’s really cool!”*
- Feels like progress. It isn’t.
- **Don’t count it.** Steer back to something specific
 - “When would you have needed this last week?”

(2) Fluff

- Sounds like: *"I usually..." "I always..." "I would..."*
- Generic claims and promises about the future
- **Anchor it** to a specific event
 - "When was the last time? What happened?"

(3) Feature requests

- Sounds like: *"It should have a button that..."*
- **Dig:** "Why do you want that?"
- The requirement is **the problem behind the request**

What's wrong with these questions?

- “Do you think .gitignore is well designed?”
- “Would you use a warning when a rule has no effect?”
- “What would your dream version do?”
- “Wouldn't it be better if ! always worked?”

Better questions

- “Tell me about the last time a file got committed that shouldn’t have.”
- “What did you do the last time a rule didn’t seem to work?”
- “What else have you tried when ignoring went wrong?”
- “If you ignore a folder but want one file from it, what do you expect?”

The requirement is in the follow-up

You: You ignore a folder but un-ignore one file in it. What happens?

Them: It should just work.

You: What does “just work” mean for that file?

Them: It shows up in git status and I can commit it.

You: Does it matter how you wrote the folder rule?

Them: No. `build/` and `build/*` mean the same thing to me.

Follow-ups to keep handy

- “Can you give me a specific example?”
- “What did you expect instead?”
- “Why do you bother?”
- “What else have you tried?”
- “What are the implications of that?”
- Silence. Count to five.

**Ask questions that
produce checkable
answers**

(1) Invariant

- **What should never happen?**
- e.g., *“Is there any file that should never end up in a commit, no matter what?”*

(2) Round-trip

- **Do it, then undo it. What do you expect?**
- e.g., *“If you add a line and then delete it, should git status look the same as before?”*

(3) Metamorphic

- **Change the input a little. What stays the same? What changes?**
- e.g., *“If you move a rule to the bottom, or into a subfolder’s .gitignore, should anything change?”*

(4) Oracle

- **How would you know it got it wrong?**
- e.g., *“How would you find out it ignored something it shouldn’t have?”*

**In recitation, the spec gave
you the properties. In P3,
users are the spec.**

Example

```
# .gitignore  
build/  
!build/keep.txt
```

- **Interviewee:**

“keep.txt gets committed. I explicitly said not to ignore it.”

What Git does

- `keep.txt` stays ignored
- Git can't re-include a file if its parent directory is excluded
- Writing `build/*` instead makes it work

Source: git-scm.com/docs/gitignore

Who to interview

- **(1) A classmate** not on your team
 - Uses opencode, doesn't know the internals
- **(2) Course staff** (sign up on the Slack sheet)
 - Knows intended behavior and edge cases
- **(3) Someone outside the course** who has used an AI coding tool
 - Least course bias
- End every interview with **"Who else should I talk to?"**

Running the interview

- **Before:** ~1-page script, feature ready to demo
- **Open:** explain the purpose, ask before recording
- **Their workflow first**, then the live demo
- **Core:** script questions + follow-ups
- **Close:** “What should I have asked?”
- **After:** clean up notes the same day

Take notes you can cite

```
Outside (data analyst) · Oct 27 · 18 min  
Q4: build/ vs build/*?  
O-11 "Same thing. keep.txt  
      should get committed."  
O-12 "If it won't work, tell  
      me. Don't silently ignore it."  
O-13 [follow-up: how would  
      you notice?] "When my  
      teammate's build breaks."  
O-14 (observed: ran git add -f)
```

- **Their words**, not yours
- **One ID per line**
 - S- staff, C- classmate, O- outside
- **Mark yourself:** follow-ups, observations
- **Record:** role, date, length, questions

From notes to requirements

ID	Evidence	Requirement	Acceptance criteria
R1	O-11, C-05	A ! rule re-includes the named file, even inside an ignored directory	Given <code>build/</code> then <code>!build/keep.txt</code> , <code>git status</code> lists <code>build/keep.txt</code> as untracked
R2	S-08, O-12	Ignoring an already-tracked file tells the user it has no effect	With <code>secrets.env</code> tracked, adding it to <code>.gitignore</code> makes <code>git status</code> print a warning naming the file
R?	—	Ignoring should be intuitive	Users like it ← not testable, not traced

P3: 5–8 requirements, new or already implemented, each with acceptance criteria and citations

Participation Activity

- Pair up.
 - First Interview Topic: **“undo”**
 - Second Interview Topic: **Copy and paste**
- Start with their life: the last time they used the feature
- Ask one question of each kind:
 - invariant,
 - round-trip,
 - metamorphic,
 - oracle
- 4 minutes each participant, then write one requirement with acceptance criteria, citing a line

Which of your questions got a checkable answer? Which questions didn't produce helpful answers?

P3 Checklist

- **This week:** declare your feature, book all three interviews
 - Prefer deterministic features over LLM, streaming, or network-heavy ones
- **Script:** ~1 page, all four question kinds + follow-ups
- **Notes:** line IDs, role, date, duration
- **Requirements:** 5–8, each with acceptance criteria and citations
- **Due Thursday, October 29th, 11:59pm**

Takeaways

- Don't blame the user. Blame the design, then go fix it.
- What people do and what they say are different evidence
- If your mom could lie to you about it, it's a bad question
- Ask about specific things that happened, not hypotheticals
- A requirement is something you can check, traced to someone who said it