



Software Testing



Show and Tell



- Take turns sharing what you built, tried, or discovered in the homework assignment.
- As you listen, ask questions, react, and share ideas. If you're not sure what to say, try:
 - ***I like...*** What stood out to you? What worked well or was interesting?
 - ***I wish...*** What could be improved, changed, or tried differently?
 - ***I wonder...*** What are you curious about? What do you want to know more about?
- You can also ask “how did you do that?” and “wait, *it can do that?*”

Outline for Today

- ***Software Testing***
 - Going on a bug hunt!
- ***Manual Testing***
 - How to look for errors by hand.
- ***Automated Tests***
 - How coding agents check their work.
- ***Issues and Bug Reports***
 - What do do when you find a bug.

Software Testing



- All nontrivial software has bugs. You've almost certainly encountered them in your regular day-to-day life.
- But what exactly *is* a bug?
- A bug occurs when
 - intended behavior* $\neq$ *actual behavior***
- Stated more simply: the program did something that wasn't what it was supposed to do.
 - (Who decides what "intended" is?)
- ***Question:*** How do we reduce the *quantity* and *seriousness* of the bugs that we ship in our code?

Software Testing

- We search for bugs by **testing** software.
As in: “the hero was *put to the test*.”
Not: “I got an 87% *on my test*.”
- An **test case** is a
 - precisely-defined scenario
 - where we run code,
 - observe what happens, and
 - compare that against what’s supposed to happen.
- Example: A test case for a calculator program:
 - Type "193 + 137" and hit "=".
 - Look at the display.
 - Confirm that you see 330.

Testing Code By Hand

Manual Code Testing

- A **manual test** is a test case where you operate the software and make sure it works as intended.
- If you observe something go wrong, congrats! You just found a bug.
- If everything seems fine, does that mean there are no bugs in the code?
- **No:** Maybe something is broken, but you didn't use the code in a way that triggered it.
 - Example: **Steve Jobs' 2007 iPhone demo**.
- Hunting for bugs is a skill you can build. And there are some good guiding principles you can follow.

Manual Code Testing

- My recommendation: ***act like a three-year-old***.
 - Start by doing the things that you're expected to do. Make sure that the "normal" path works correctly.
 - Then push the limits. Try things you're not supposed to do and make sure the software responds appropriately. You shouldn't be able to break anything.

"A software tester walks into a bar. Runs into a bar. Crawls into a bar. Dances into a bar. Flies into a bar. Jumps into a bar. And orders: a beer. 2 beers. 0 beers. 999999999 beers. a lizard in a beer glass. -1 beer. "qwertyuiop" beers."

- ***Source***

Manual Code Testing

- **Pros:**
 - If you see the software do something unintended, you've definitely witnessed a bug.
 - Manual review can also surface areas for future improvement: quality-of-life issues, etc.
- **Cons:**
 - **Labor-intensive.** You have to test code every time you make a code change (details later). And in complex software, you can't test every feature by hand.
 - **Out of the agentic loop.** Coding agents work best when they can observe and correct their own mistakes. Manual testing breaks that loop.
- Manual testing is a necessary component of testing, but it can't be your "first line of defense."

Automated Testing

Automated Testing

- **Idea:** Have the computer automatically perform the checks we'd do by hand.
- To test an existing piece of code, write *new code* that executes the code that needs to be tested and checks the result.
- **Good News:** Coding agents are getting pretty good at writing their own tests.
 - But they aren't perfect! You often need to guide them.
- What does that look like?

Demo: ***Phoenician Match Game***

What Tests Do We See?

- ***Unit Tests***
 - Check individual functions / modules / etc. of the full system to make sure each one works in isolation.
 - Excellent at detecting bugs in individual pieces of code.
 - Excellent in localizing where errors are.
 - Can't catch everything...

What Tests Do We See?

- ***Smoke Tests***

- Run the full system and do basic checks to see that nothing is “obviously” wrong.
- Name comes from electrical engineering: if you plug the device in and smoke comes out of it, something is wrong!
- Useful for quickly validating that a code change didn't break anything.

What Tests Do We See?

- ***End-to-End (e2e) Testing:***
 - Drive the full piece of software, controlling the inputs, and make sure that the outputs are correct.
 - Excellent at determining when the full system has issues.
 - Not the best at *localizing* those issues: you know something is broken, but you don't know where.
- ***Headless Test:*** Run a website in an invisible web browser with the computer controlling the mouse and keyboard.
 - The development environment ships with ***Playwright***, a tool for writing and running these kinds of tests.

Time-Out for Announcements!



Flu and COVID Shots!



- Stanford offers flu and COVID shots to students.
- There's an event on Tuesday, October 6 where you can get your vaccines. Use *[this link](#)* for details.
- Protect yourself and the people around you. Yay!

Assignment 3

- Assignment 3 goes out today. It's due next Tuesday at 1:00PM.
 - ***Go on a bug hunt!*** We give you buggy code, you figure out what's wrong and fix it.
- For those of you who recently joined the course:
 - Please complete A0, A1, and A2 as soon as you can.
 - Come talk to us if you have questions on the material and want to get caught up.

Tutorial Sessions

- It was great meeting you at tutorial sessions this week! We're amazed by your creativity.
- ***Friendly reminder:*** tutorial sessions are mandatory; 40% of your course grade is determined by participation.
- ***For next week:*** Study up on core terminal commands and git basics, and be prepared to drive the computer from the terminal.

Back to CS193V!

When You Find A Bug

“But the Tests All Passed!”

- If any test cases fail, there’s probably a bug in your code.
 - Or the test case *itself* is buggy... 
- If all the tests all *pass*, it doesn’t mean your code is bug-free. It just means you didn’t see any bugs.
- It is *normal* and *common* to find bugs in code even when all the tests pass.

A Normal Workflow

- A typical workflow with a coding agent goes like this:
 - You ask it to do something.
 - It comes up with some code and some tests.
 - It runs the tests, fixes bugs it finds, and repeats until it works.
 - You look at the finished product.
 - You find a bug that the tests missed.
- What should you do next?

Demo: ***Mental Blocker***

Go on a Bug Hunt!

- Get the game files from GitHub:
`git clone https://github.com/cs193v/mental-blocker`
- Navigate in your file browser (Finder, File Explorer, etc.) to your projects folder.
- Find the mental-blocker directory. Double-click index.html.
- **Goal:** Find as many bugs as you can!

Reporting a Bug

- When reporting a bug, provide all of the following information:
- ***Steps to Reproduce***
 - Find a *reliable* way to trigger the behavior, then describe in detail how to do it. Be specific.
- ***Expected Behavior***
 - Describe what you expected to happen. Be specific!
- ***Actual Behavior***
 - What *actually* happened? Be specific. Include screenshots, error logs, etc. as appropriate.
- This is true for bugs both in software written by humans and written by machines.

GitHub Issues

- You can create *issues* on GitHub to report bugs, request new features, etc.
- Each issue gets an assigned number (e.g. #137) that you can use as a shorthand.
- Coding agents can read issues you post on GitHub, and they can file issues for you as well.
- If you push a commit whose message says “fixes #NN,” GitHub automatically marks the issue as resolved.

Keeping Your Code Clean

Avoiding Regressions

- A **regression** occurs when changing code to fix a bug results in another (possibly unrelated) bug.
 - The fix may itself be wrong and cause other knock-on effects.
 - The fix may be correct, but expose a second, previously unknown bug whose behavior was masked.
- Takeaway: Don't just tell the LLM to fix things and call it a day! You need to make sure that it really fixed things without introducing regressions.
- How do you do this?

The “Red-First” Workflow

- Ask your coding agent to fix issues *red-first*. Here's what that means:
 - Before starting on any work to fix the bug, the coding agent writes a test case that fails (“goes red”) due to the existing bug.
 - The coding agent then attempts to fix the bug, using the test case it wrote to validate that the fix worked.
 - The coding agent then saves that test as part of the permanent test suite to ensure the bug can't regress (come back) later.
- Afterwards, always ask the coding agent to run the full test suite before declaring the bug fixed.
 - It probably will do this anyway, but sometimes you need to prod it a bit. 😊

What else can we do to hunt bugs?

Further Techniques

- In a **code review**, a programmer other than the code author reads code before that code is committed into the repo.
 - Claude Code can do this for you: type `/code-review` and hit enter. (Uses lots of tokens!)
- Many systems use **fuzz testing**, software that runs randomly-generated inputs through a program to make sure the program doesn't misbehave.
- There are **static analysis** tools that can automatically find bugs in code without running that code; this is an active area of research.
- Companies often release **nightly** or **preview** versions of code to volunteers to try things out and spot bugs before a full rollout.

Next Time

- ***Design for Vibecoders***
 - Guest lecture series by the amazing Christina Wodtke!