# Software Archaeology and Anthropology

17-313 Fall 2024

Foundations of Software Engineering

https://cmu-17313q.github.io

Eduardo Feo Flushing

# Administrivia

- Slack
  - Please add a profile picture.
  - Ask questions in `#general` or `#technical-questions`.
    - Please use threads.
    - Use the search tool
- Office hours can be found on the course home page: http://cmu-17313q.github.io

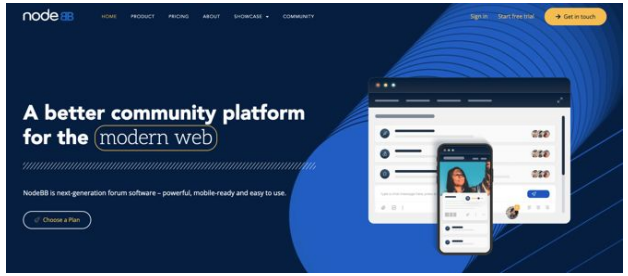
# Smoking Section

- Last two rows

# Team Formation Survey due Thursday

- Form groups based on schedule availability.
  - This is ridiculously important.
  - Identify experience and working styles.
  - Participation point
- Google Form, posted on slack

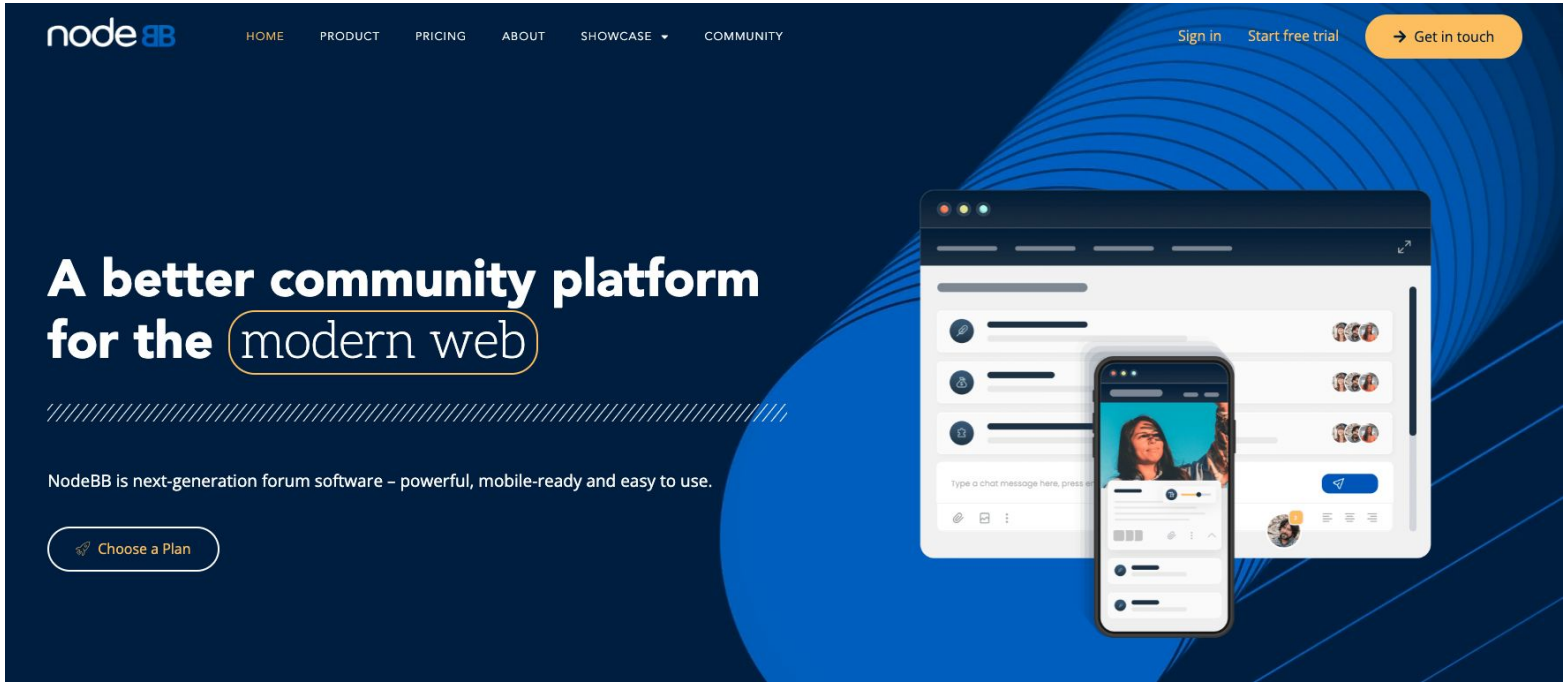
# Project P1




- **P1A**: Checkpoint due next Sunday (September 1st)
  - Only 5% of total P1 points – meant to ensure you start on time
- **P1B**: Due Thursday next week, September 5th )
  - Refactor a javascript file to improve its quality
  - It will be posted tomorrow
  - Start early

# Context: big old pile of code

- … do something with it!



# You will never understand the entire system!

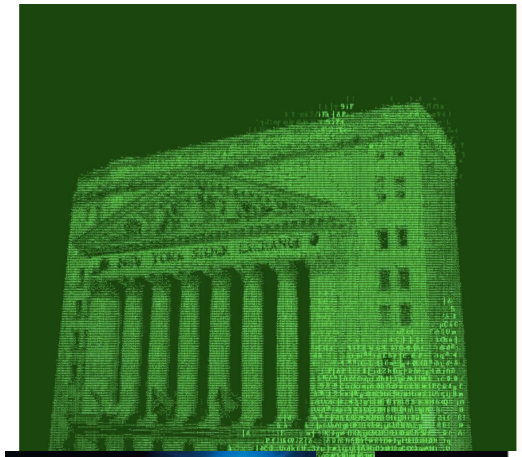
# Challenge: How do I tackle this codebase?

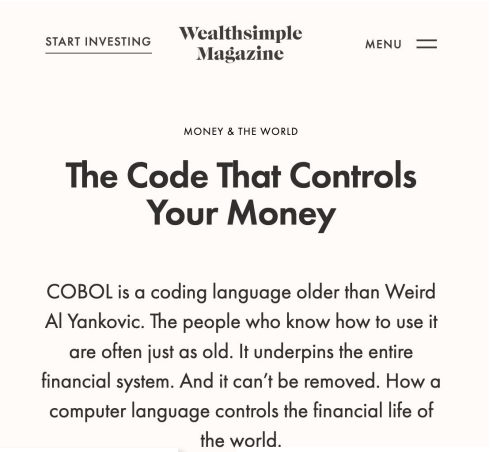
# Challenge: How do I tackle this codebase?

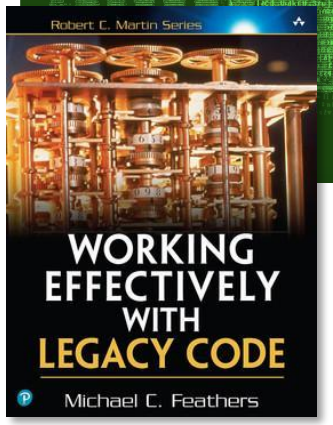
- Leverage your previous experiences (languages, technologies, patterns)
- Consult documentation, whitepapers
- Talk to experts, code owners
- Follow best practices to build a working model of the system

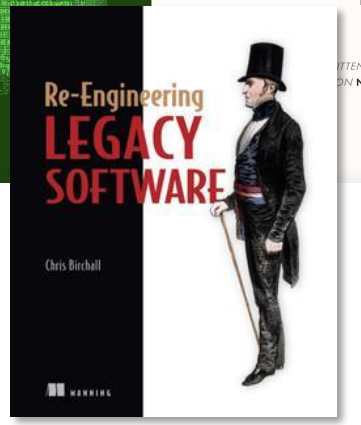
# Bad news: There are few helpful resources!

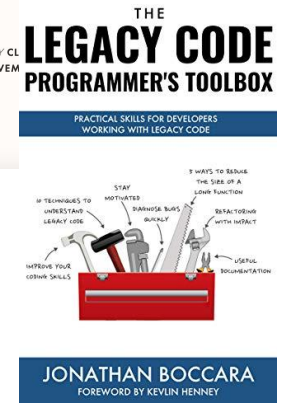
- **Working Effectively with Legacy Code.**
  Michael C. Feathers. 2004.
- **Re-Engineering Legacy Software.**
  Chris Birchall. 2016.
- **The Legacy Code Programmer's Toolbox.**
  Jonathan Boccara. 2019.



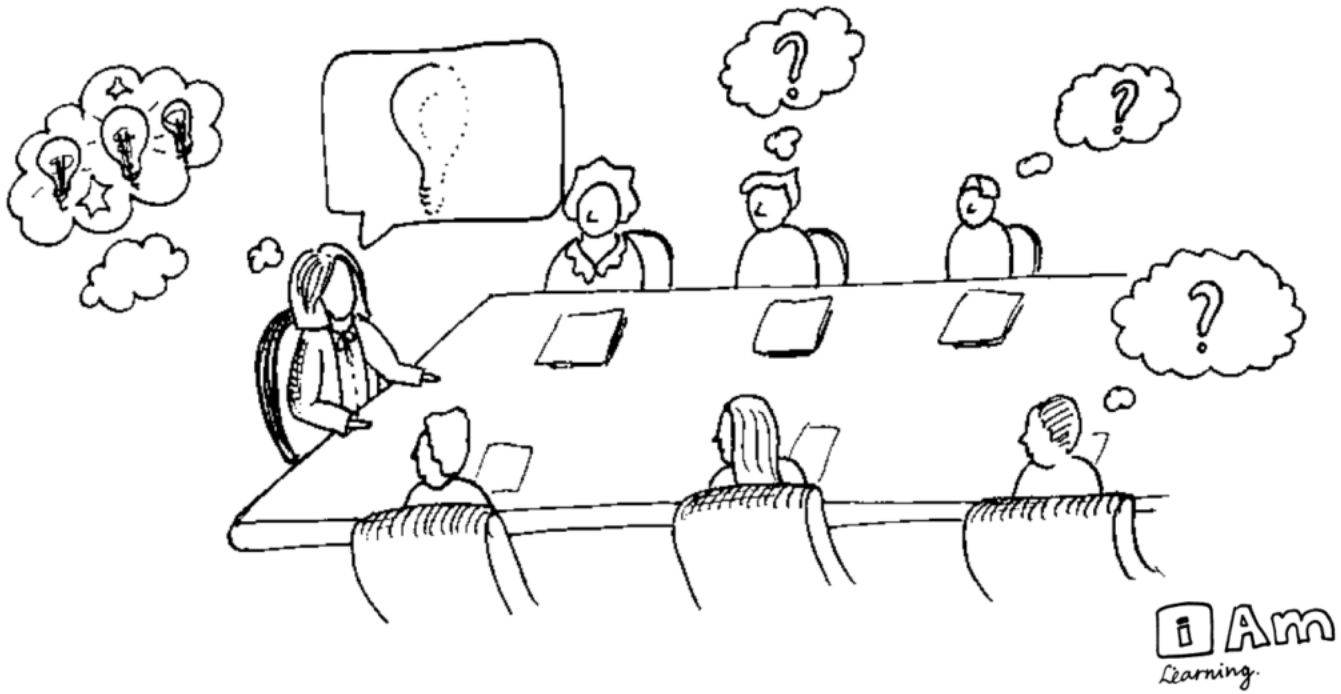


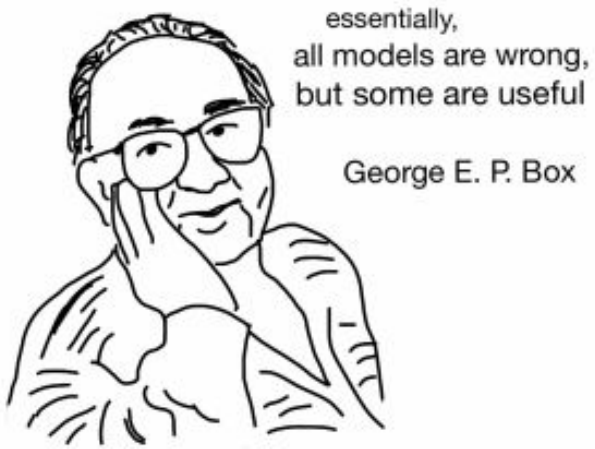



# Why? Because of Tacit Knowledge

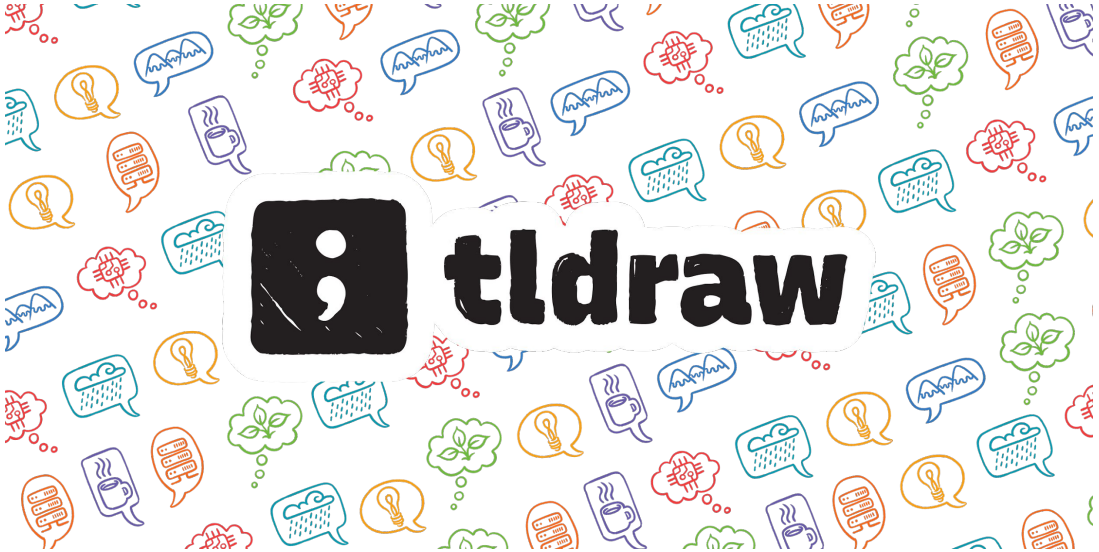


# Today: How to tackle codebases

- Goal: develop and test a working model or set of working hypotheses about how (some part of) a system works
- Working model: an understanding of the pieces of the system (components), and the way they interact (connections)
- Focus: Observation, probes, and hypothesis testing
  - Helpful tools and techniques!



# Live Demonstration: tldraw




https://github.com/tldraw/tldraw

# Steps to Understand a New Codebase

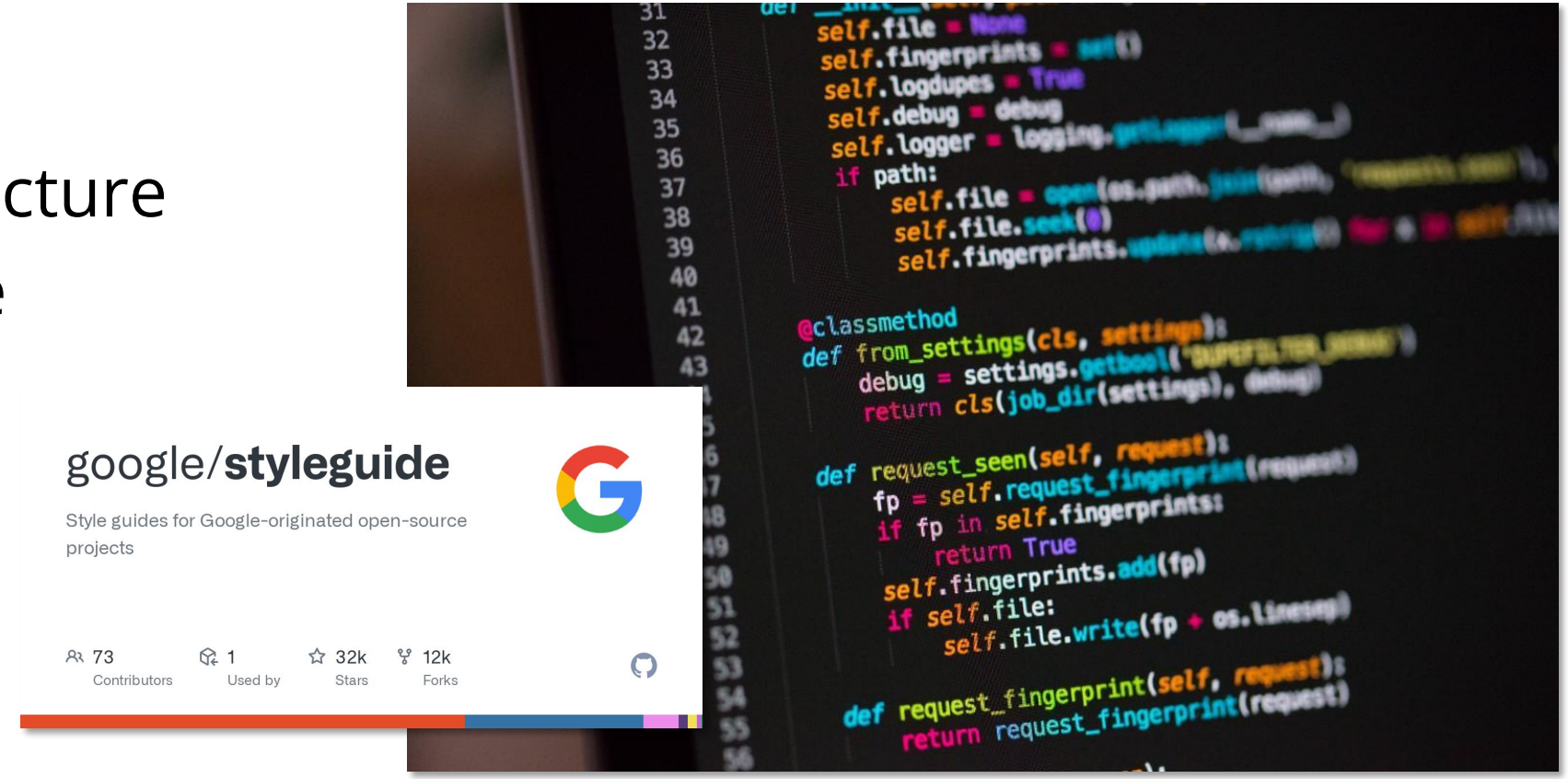
- Look at README.md
- Clone the repo.
- Build the codebase.
- Figure out how to make it run.
- What do you want to mess with?
  - Clone and own
- Traceability - Attach a debugger
  - View Source
  - Find the logs.
  - Search for constants (strings, colors, weird integers (#DEADBEEF))

# Participation Activity

- Take out a piece of paper.
- Write down one pro and one con about trying to understand a new codebase by compiling and building it vs. just reading the code.
- Pair with your neighbor and discuss your answers. Do you agree?
- Share with the class!
- Write your own andrewID on the paper, leave it at the end of class.

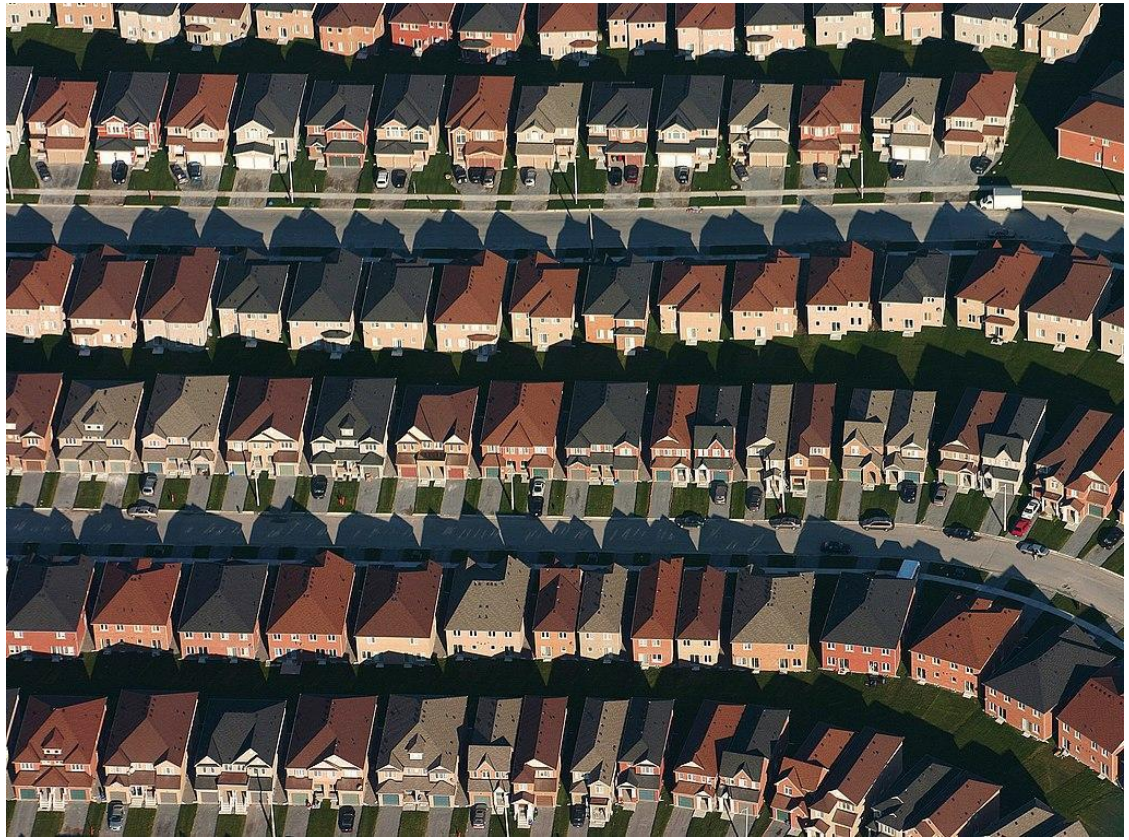
# Observation: Software is full of patterns

- File structure
- System architecture
- Code structure
- Names
- …

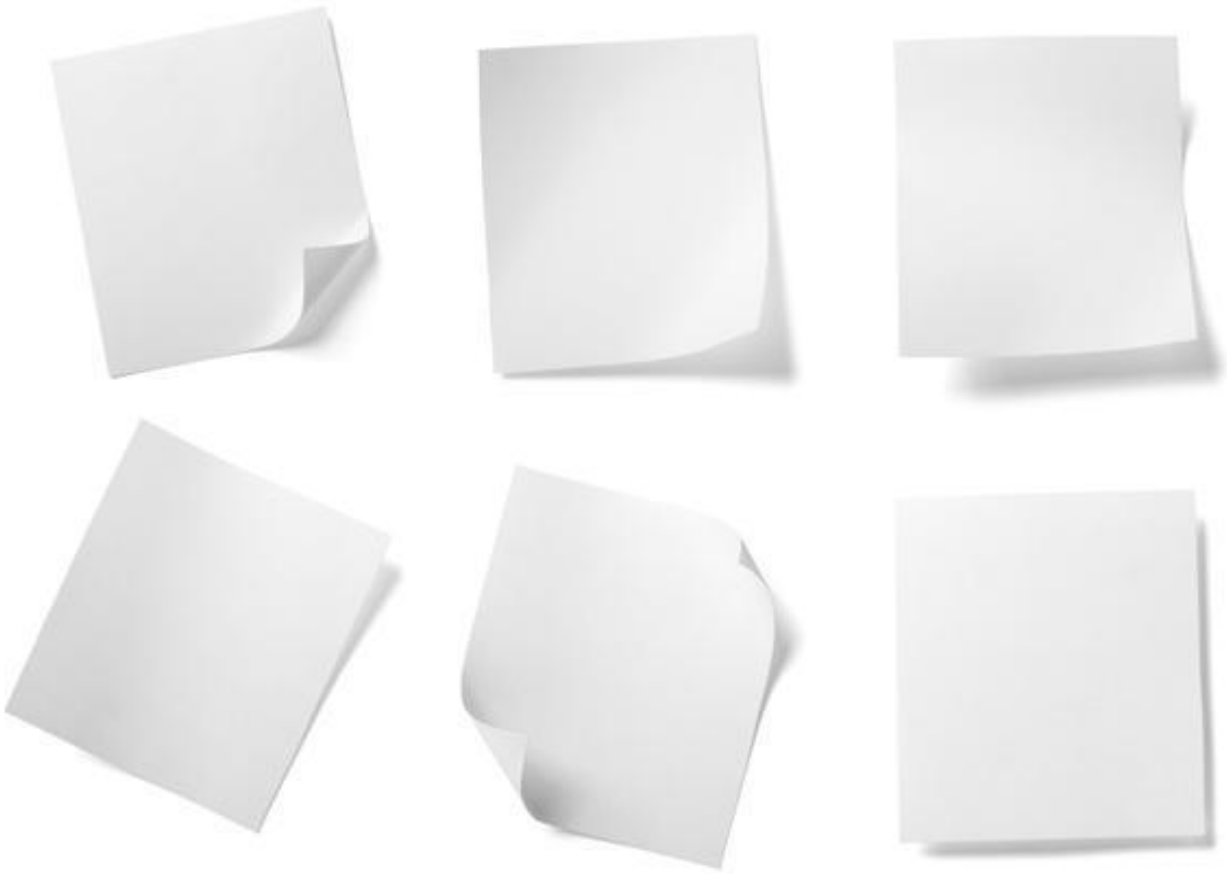


# Observation: Software is massively redundant

- There's always something to copy/use as a starting point!



# Observation: Code must run to do stuff!



# Observation: If code runs, it must have a beginning…

# The Beginning: Entry Points

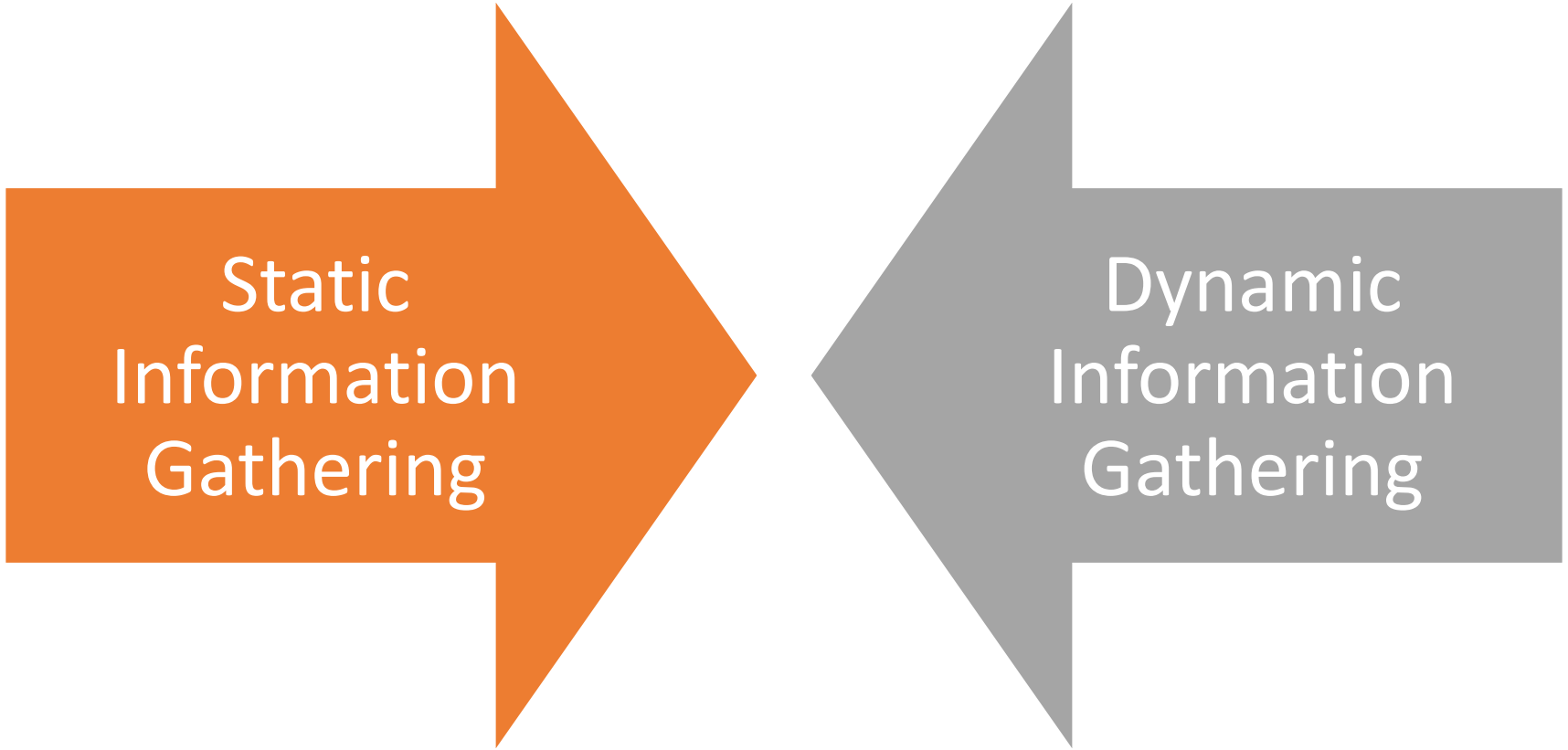
- Locally installed programs: run cmd, OS launch, I/O events, etc.
- Web apps server-side: Browser sends HTTP request (GET/POST)
- Web apps client-side: Browser runs JavaScript, event handlers

# Can running code be Probed/Understood/Edited?

| Transparent | Translucent | Opaque |
|---|---|---|
| Source code built locally | Binaries running locally | Server-side apps running remotely |
| | Open source / Closed source | Open source / Closed source |
| (P+U+E) | (P+U) / (P) | (U) / (Talk to NSA) |

# Creating a model of unfamiliar code

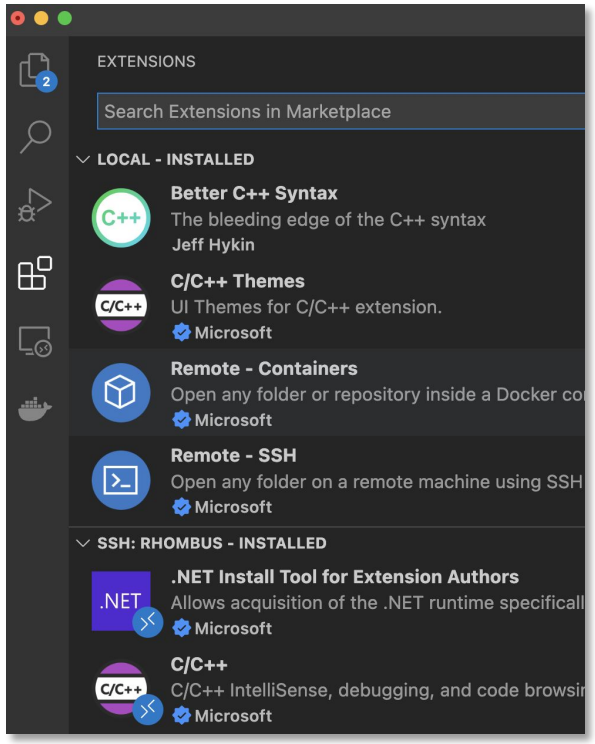
Source code built locally

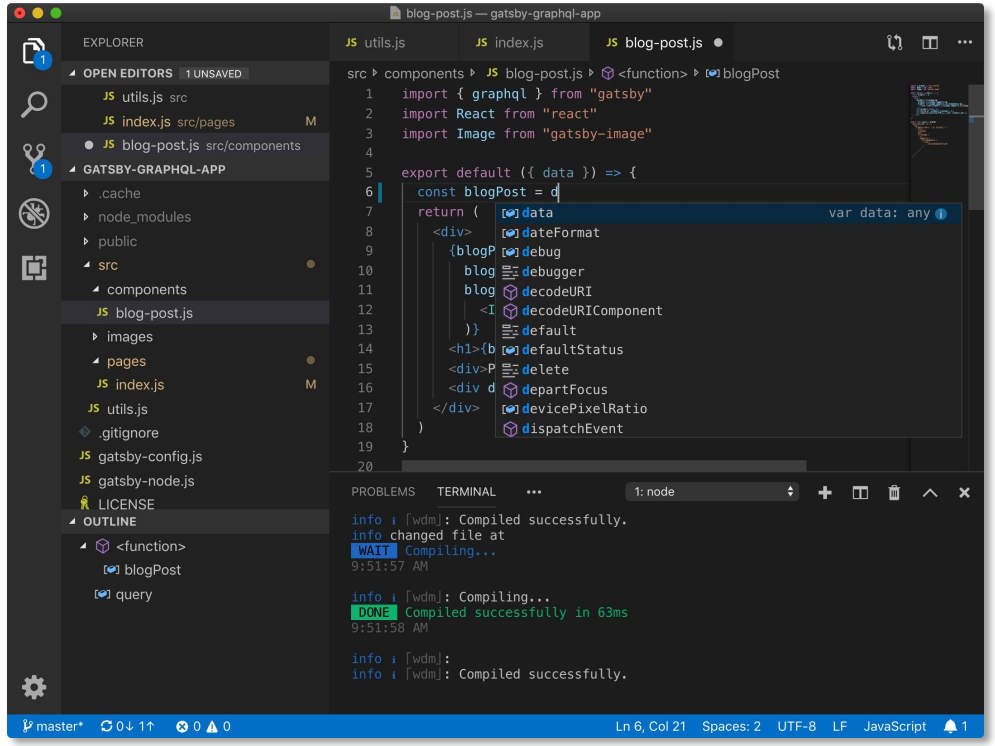

Static Information Gathering
Dynamic Information Gathering

# Static Information Gathering

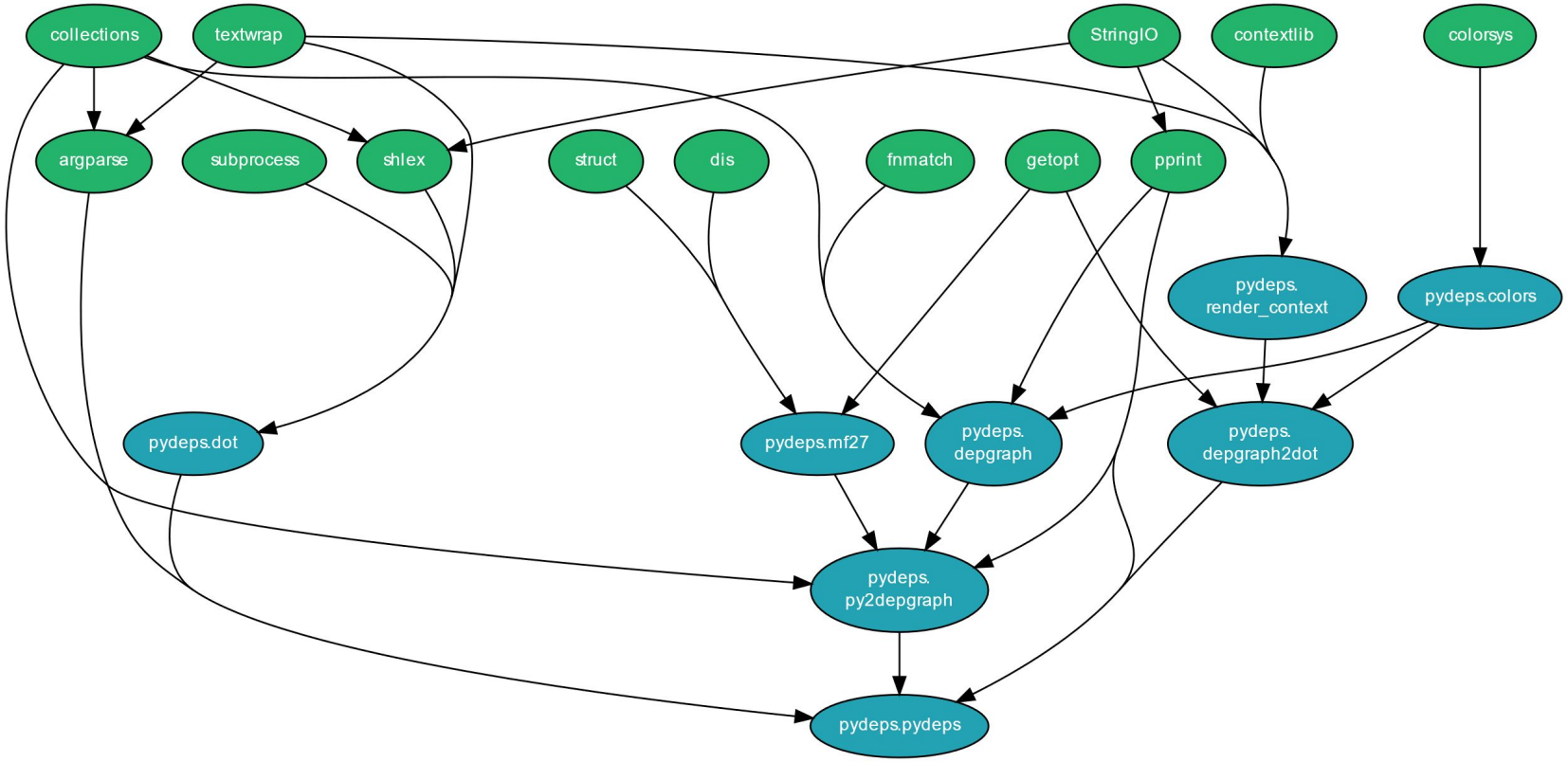
- Basic needs:
  - Code/file search and navigation
  - Code editing (probes)
  - Execution of code, tests
  - Observation of output (observation)
- Many choices here on tools! Depends on circumstance.
  - grep/find/etc.  Knowing Unix tools is invaluable
  - A decent IDE
  - Debugger
  - Test frameworks + coverage reports
  - Google (or your favorite web search engine)
  - ChatGPT or LaMA

# Static Information Gathering: Use an IDE!
# Real software is too complex to keep in your head



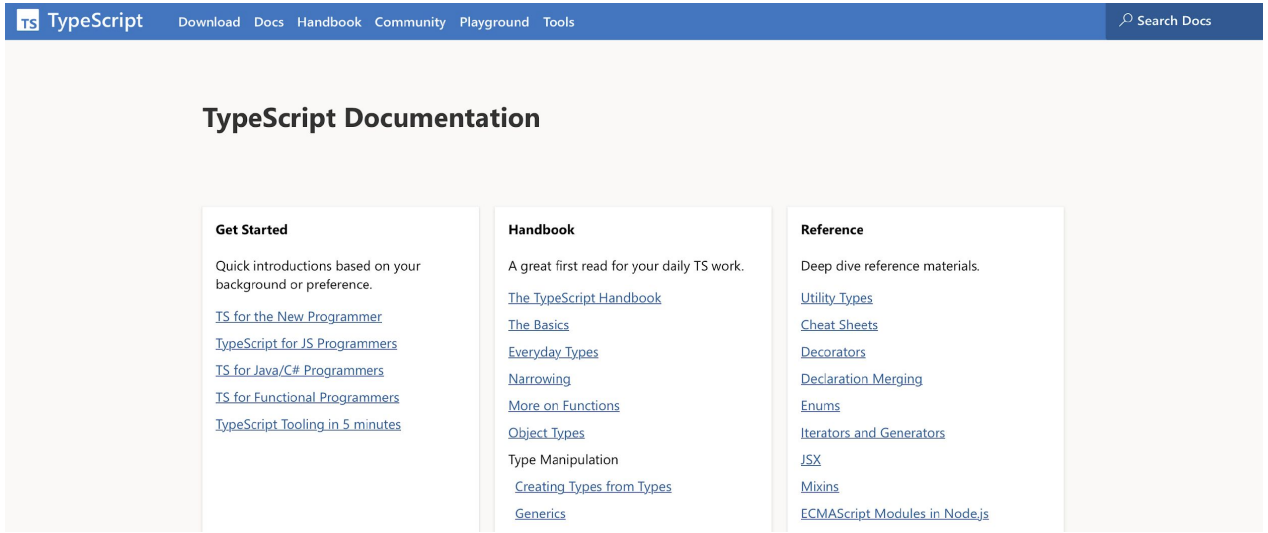



# Dependency maps

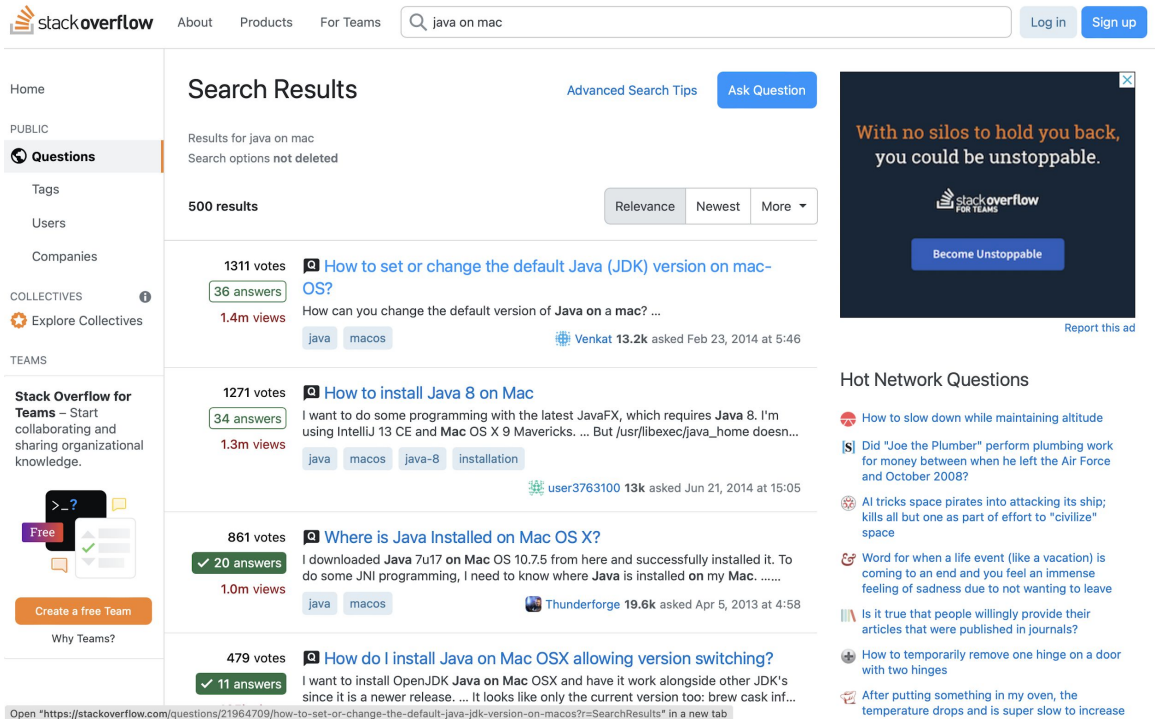


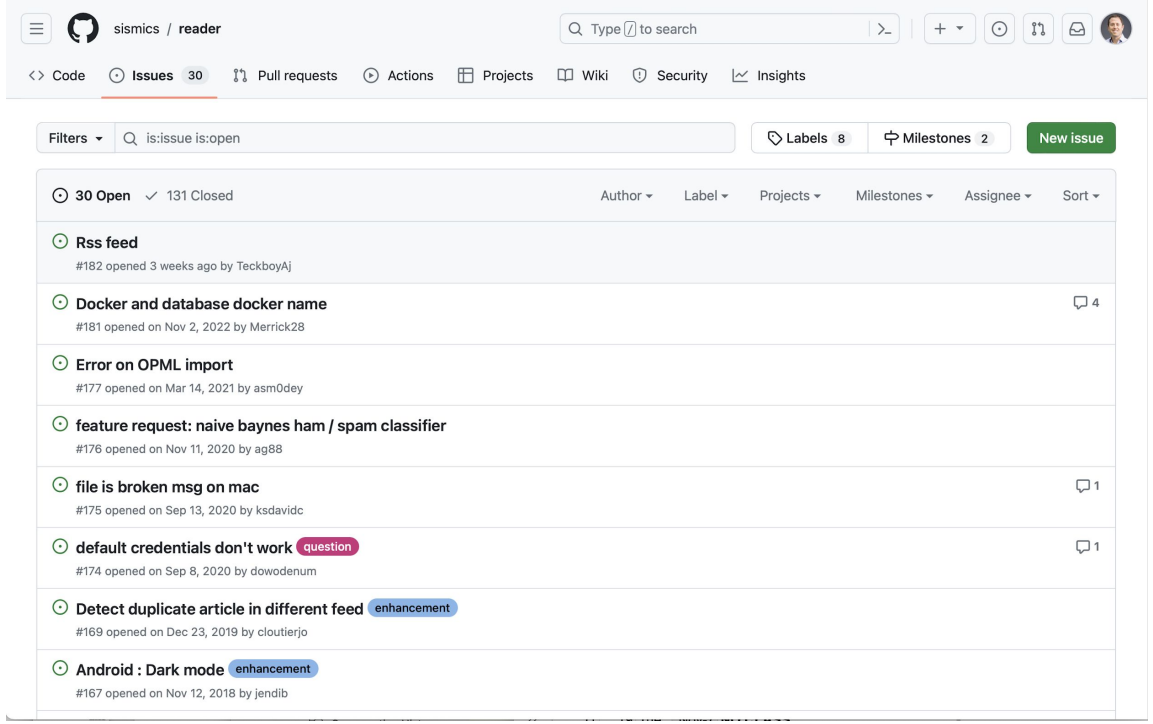
# Consider documentation and tutorials judiciously

- Great for discovering entry points!
- Can teach you about general structure, architecture (more on this later in the semester)
- Often out of date.
- As you gain experience, you will recognize more of these, and you will immediately know something about how the program works
- Also: discussion boards; issue trackers



# Discussion Boards and Issue Trackers



# Dynamic Information Gathering

## Change helps to inform and refine mental models

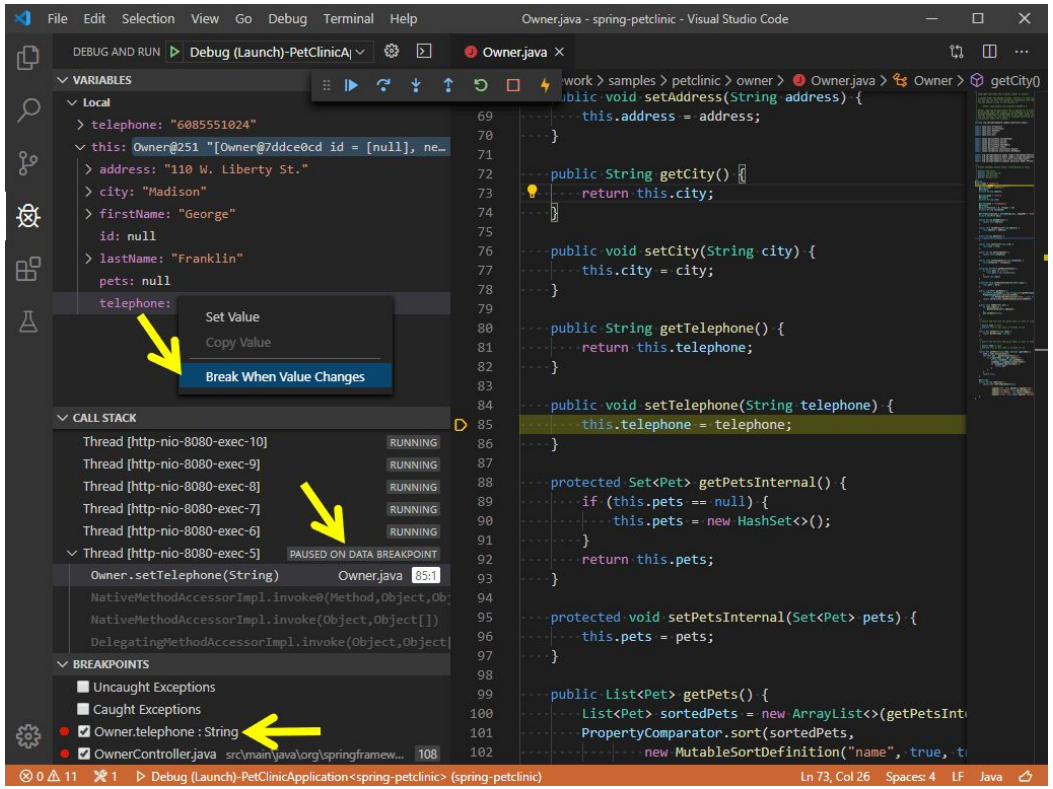
- Build it.
- Run it.
- Change it.
- Run it again.
- How did the behavior change?

# How to start?

- Confirm that you can build and run the code.
  - Ideally both using the tests provided, and by hand.
- **Confirm that the code you are running is the code you built!**
- Confirm that you can make an externally visible change
- How? Where? Starting points:
  - Run an existing test, change it
  - Write a new test
  - Change the code, write or rerun a test that should notice the change
- Ask someone for help

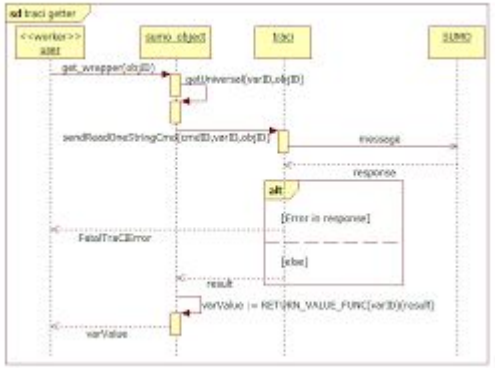
# Probes: Observe, control or “lightly” manipulate execution

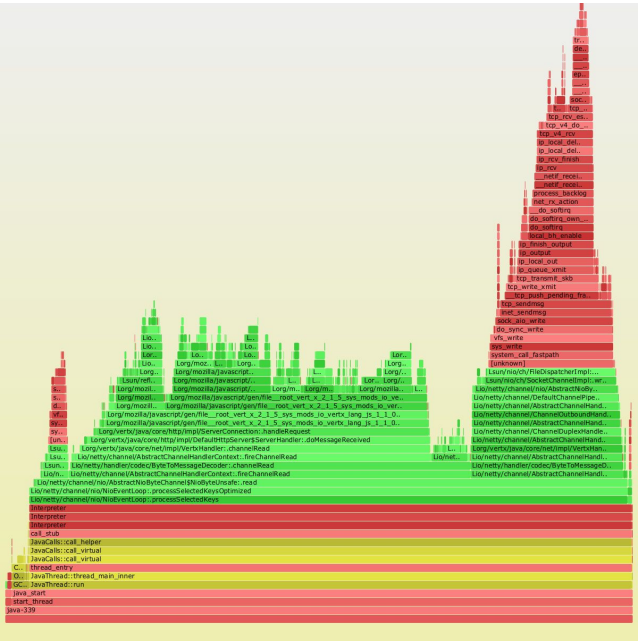
- `print(“this code is running!”)`
- Structured logging
- Debuggers
  - Breakpoint, eval, step through / step over
  - (Some tools even support remote debugging)
- Delete debugging
- Chrome Developer Tools



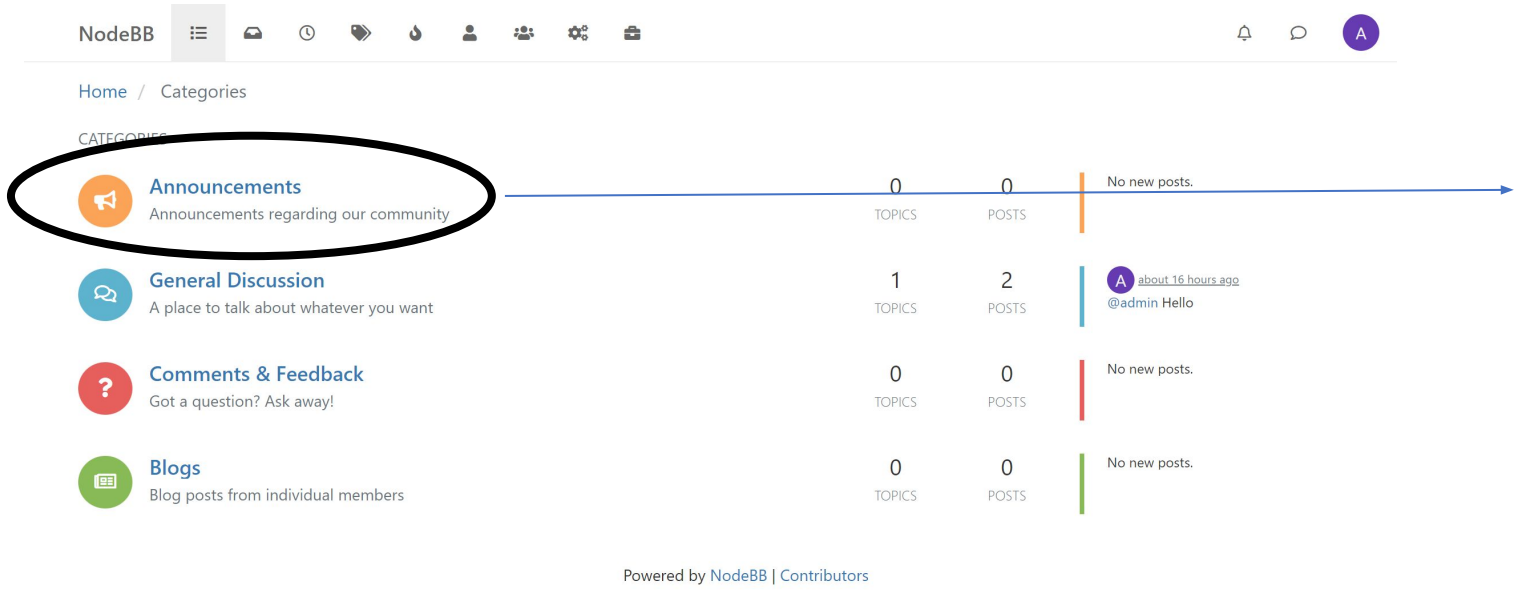
# Runtime code analysis tools

- Collect runtime traces and visualize them
  - Flame graphs
  - Sequence diagrams
- Use judiciously



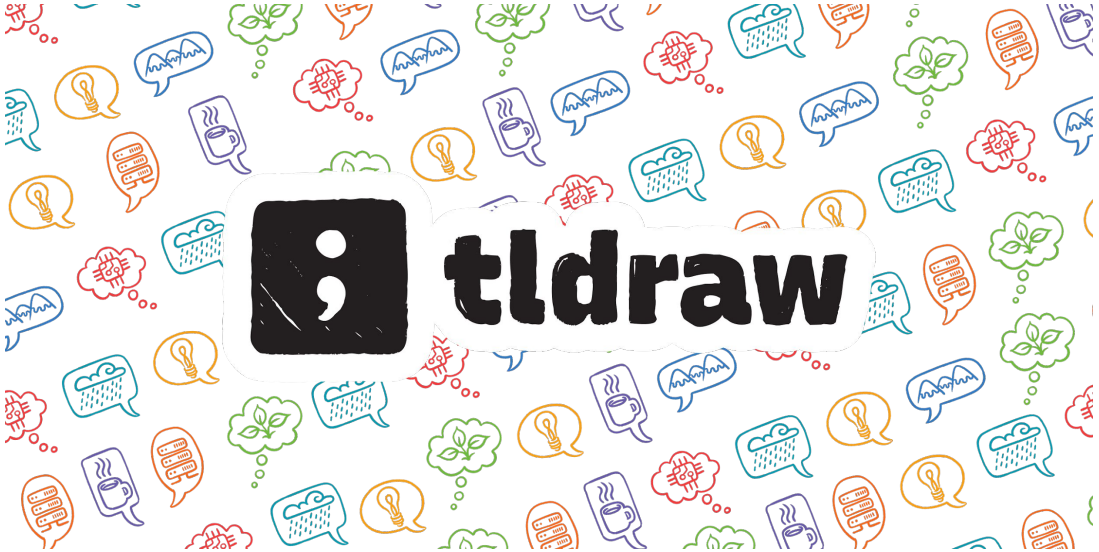



# Tip: Find a particular thing and trace the action backward




E.g.,
Where do categories come from?
How are they stored?
How are they rendered?

# Let’s try some of these techniques again…




https://github.com/tldraw/tldraw

# Remember…

- Reading and understanding code is one of the most important skills you should learn
- It's common to get stuck or feel overwhelmed. **Don't give up!**
- Consider yourself lucky! Things are much easier today

# Learning Goals

- Understand and scope the task of taking on and understanding a new and complex piece of existing software
- Appreciate the importance of configuring an effective IDE
- Contrast different types of code execution environments including local, remote, application, and libraries
- Enumerate both static and dynamic strategies for understanding and modifying a new codebase