About me

- BEng Computer Engineering (Singapore)
- Platoon Instructor (Singapore Armed Forces)
- PhD CompSci (St Andrews)
  - 10 Published papers
- Interned in Deutsch Telekom (Berlin)
- Early career in 3 startups (London)
  - Failedx2
  - Quitx1
- DevOps Engineer@Sky (London)
- Short stint as a Contractor (Perth)
- Stay at home dad (2020-2022)

Gource

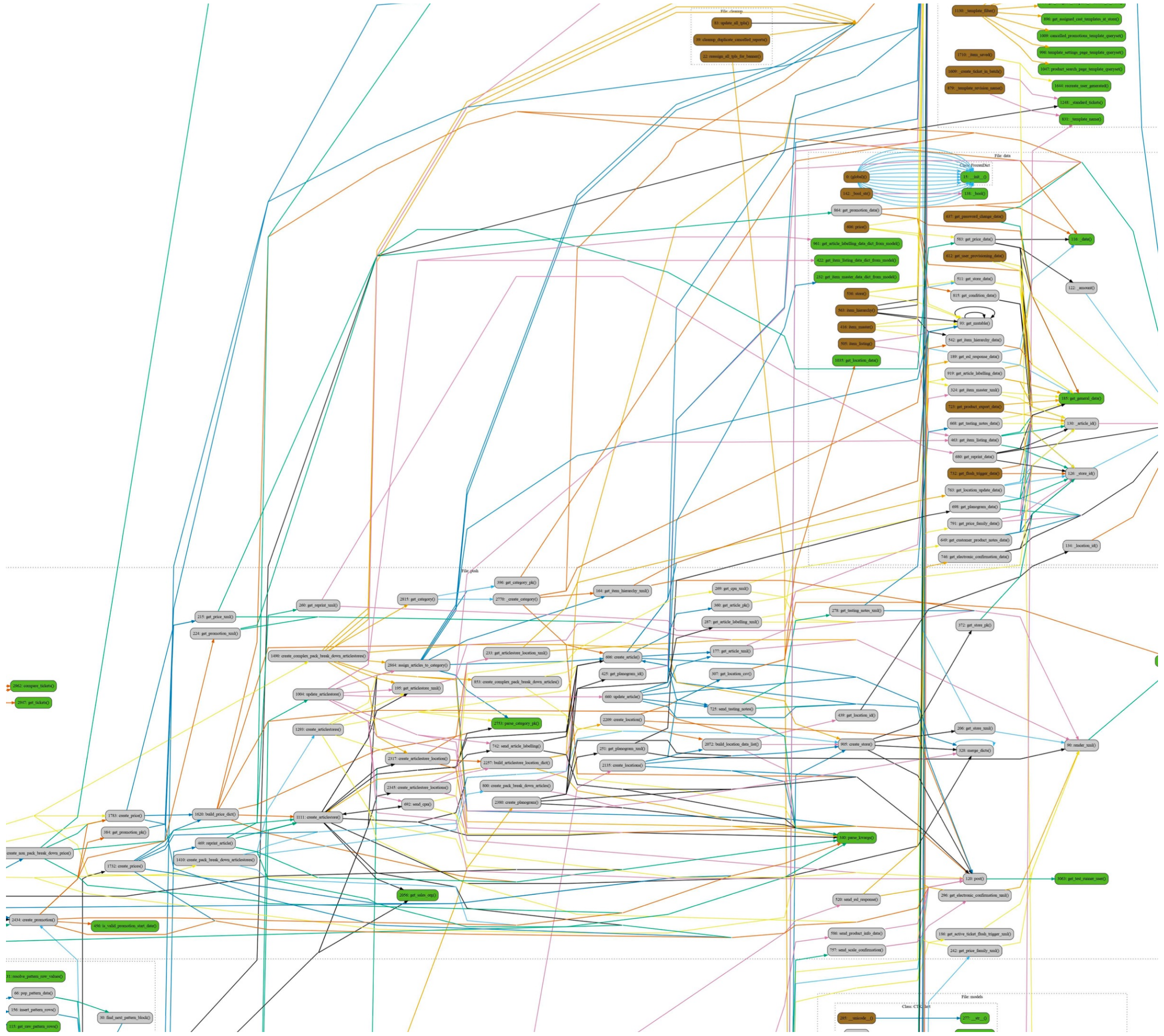
- https://gource.io

- Software to visualize version control repositories

Code2Flow

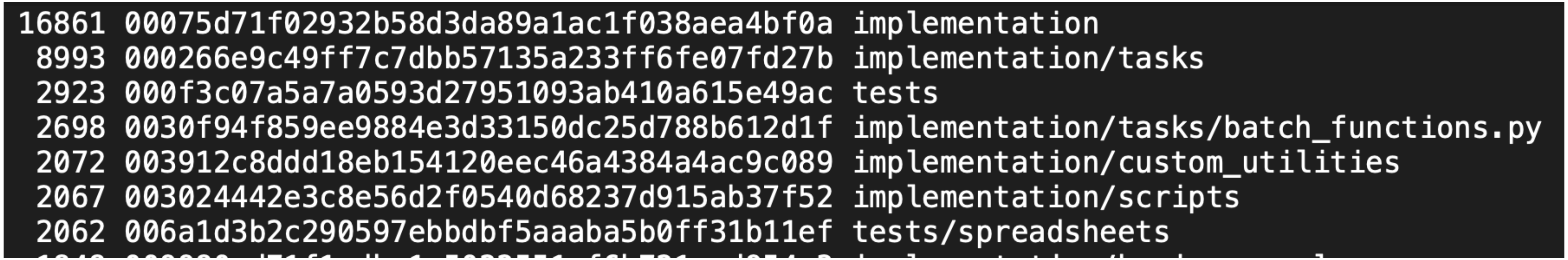
- https://github.com/scottrogowski/code2flow
- Generates call graphs for dynamic programming languagess

Code2Flow



Git Commits per file

- git rev-list --objects --all | awk '$2' | sort -k2 | uniq -cf1 | sort -rn | head



```
16861 00075d71f02932b58d3da89a1ac1f038aea4bf0a implementation
 8993 000266e9c49ff7c7dbb57135a233ff6fe07fd27b implementation/tasks
 2923 000f3c07a5a7a0593d27951093ab410a615e49ac tests
 2698 0030f94f859ee9884e3d33150dc25d788b612d1f implementation/tasks/batch_functions.py
 2072 003912c8ddd18eb154120eec46a4384a4ac9c089 implementation/custom_utilities
 2067 003024442e3c8e56d2f0540d68237d915ab37f52 implementation/scripts
 2062 006a1d3b2c290597ebbdbf5aaaba5b0ff31b11ef tests/spreadsheets
```

- batch_functions.py is 3254 lines long

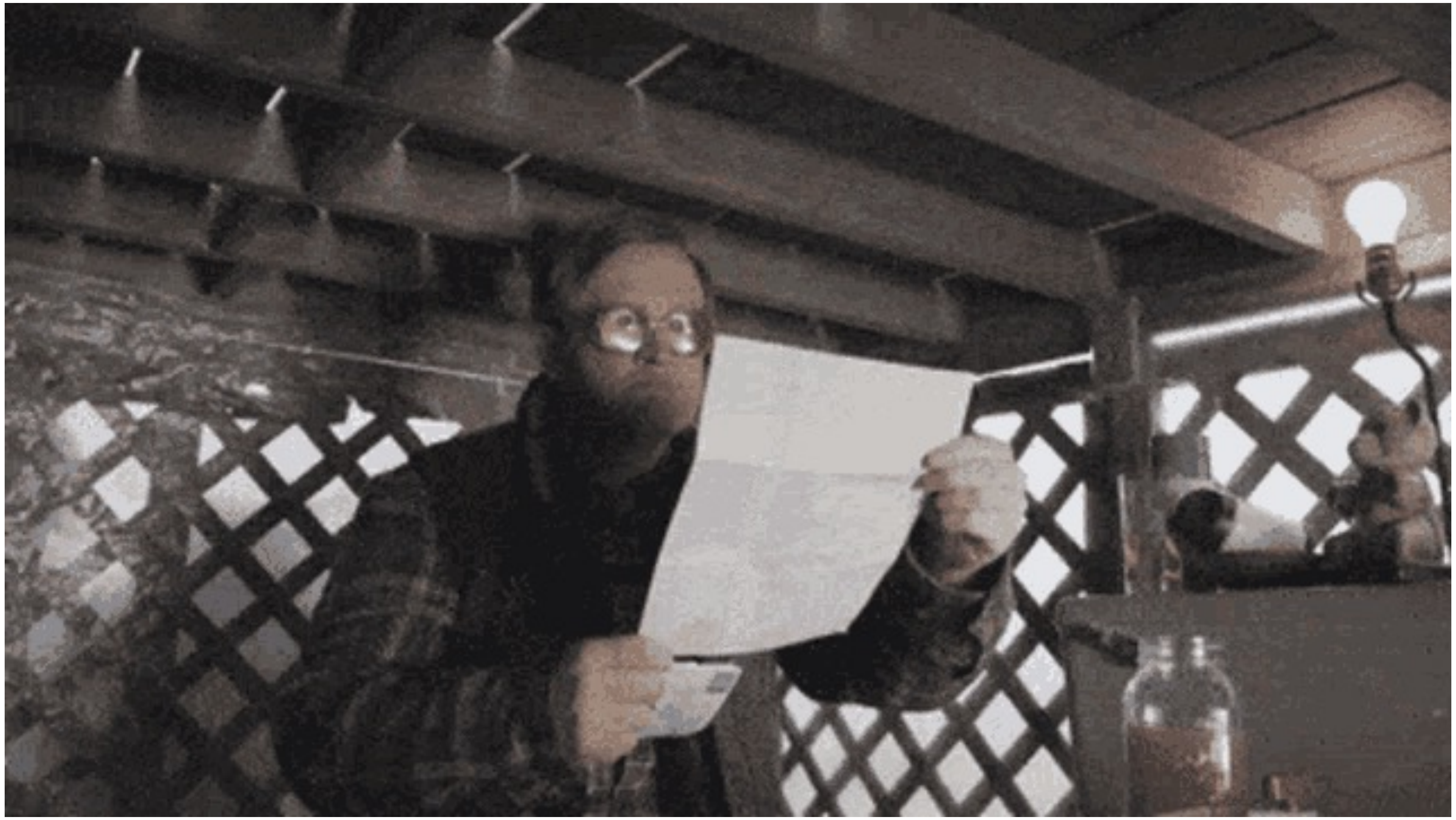
Where we are

- Complex test setup
- Slow tests
- Tangled code

Where I think we should be

- Speedy micro tests
- Untangled code
- Clear architecture

From this



To this

Before we begin..

- *“It’s important to build empathy for the code by asking ourselves to identify what has changed since it was written. If we make an effort to seek the initial good, we gain an appreciation for the pitfalls the original solution avoided, the clever ways it might have dealt with a set of constraints, and produce a refactored result that captures all these insights. “ Refactoring at Scale*

Test Driven Development

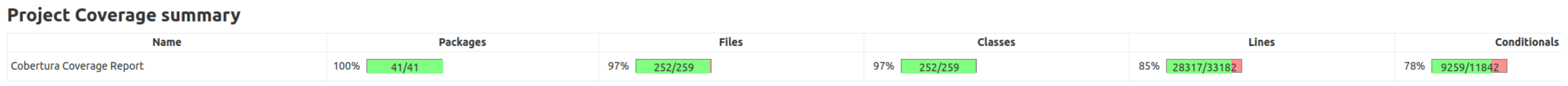
Why Tests?

- Make sure the code does what you intended

How do we write tests?

- Failing test (red)
- Passing test (green)
- Refactor (write code without changing functionality)

But we have tests!



Project Coverage summary

Name	Packages	Files	Classes	Lines	Conditionals
Cobertura Coverage Report	100% 41/41	97% 252/259	97% 252/259	85% 28317/33182	78% 9259/11842

Integrated Tests ~~are~~ can be a Scam[1]

- When unit tests are difficult to write and ineffective at catching bugs
- We write more integrated tests and less unit tests
- At best this results in tangled code
- At worst this results in more bugs
- Because of tangled code, its harder to write unit tests
- We write more Integrated tests and less unit tests

- 1. Integrated Tests are a Scam by JB Rainsberger

Integrated Tests ~~are~~ can be a Scam[1]

- When unit tests are difficult to write and ineffective at catching bugs
- We write more integrated tests and less unit tests
- At best this results in tangled code
- At worst this results in more bugs
- Because of tangled code, its harder to write unit tests
- We write more Integrated tests and less unit tests

I’m definitely not saying Integrated tests are bad but …

It’s a scam because the “solution” is actually making the problem worse.

- 1. Integrated Tests are a Scam by JB Rainsberger

How do we write tests?

- Failing test   (red)
- Passing test (green)
- Refactor   (write code without changing functionality)

- We need to spend more time listening to the feedback from the code and tests
- Iterative Refactoring should be supported and encouraged

How to Refactor - overview

- The passing tests are our safety net
- The implementation of the test and feature itself is the feedback to the current design
- We refactor feature and tests to improve design and GET MORE FEEDBACK
- Rinse and repeat

Talk is cheap

Gilded Rose kata:

https://github.com/emilybache/GildedRose-Refactoring-Kata

An open attempt:

https://github.com/DevanR/gilded_rose

I’ve included a log of my real time thoughts without any edits.

And I’ve added a new requirement. (hint: Promotions.) I have no idea how to do but will be attempting to gain insight with TDD.

What TDD will give you

- A suite of fast unit tests, which act as safety net
- Immediate feedback to improve your code design
- A method to build better design, in small, safe steps.