DEV ACADEMY
TE KURA HANGARAU
O AOTEAROA

Testing! Vitest and TDD

Week 1, Day 4

Agenda for Today

- Why test code?
- Types of tests
- Unit Testing with Vitest
- Test Driven Development (TDD)
  - Red, Green, Refactor

Why test things?

- Code quality
- Confidence
- Documentation

Code Quality

Maintainability and Simplicity

- Testable code is *generally* simpler code
- Testable code is *generally* more modular/reusable code

Confidence

- Confidence that your code completes the task it was designed to do
- Confidence that your code *continues* to complete the task it was designed to do when making changes (refactoring)

Documentation

- Lets other developers know how the code is *supposed* to work
  - Inputs, outputs, side effects
- Documents both the general use cases but also the **edge cases**

Types of Tests

THE FOUR TYPES OF TESTS

End to End

A helper robot that behaves like a user to click around the app and verify that it functions correctly.

Sometimes called "functional testing" or e2e.

Integration

Verify that several units work together in harmony.

Unit

Verify that individual, isolated parts work as expected.

Static

Catch typos and type errors as you write the code.

- End to End tests (`Selenium`, `Cypress`, `Playwright`, etc)
- Integration tests (`Vitest`)
- Unit tests (`Vitest`, `Jest`)
- Static analysis (`ESLint`, `TypeScript`)

Source: https://kentcdodds.com/blog/the-testing-trophy-and-testing-classifications

Unit Testing with Vitest

A primer on unit testing

Anatomy of a Test

Primer

```
import { numberToAccountingString } from './accounting.js'
import { test, expect } from 'vitest'

test('given the number 5, numberToAccountingString should return "5.00"', () => {
  // Arrange
  const input = 5
  const expectedOutput = '5.00'

  // Act
  const actual = numberToAccountingString(input)

  // Assert
  expect(actual).toBe(expectedOutput)
})
```

- Importing the function under test (`numberToAccountingString`)
- Importing the necessary methods from the `vitest` library
- `test` block to describe a single test.
- `// Arrange`
  - Setup the test data
  - Setup the expected output
- `// Act`
  - Call the function under test to get the actual output
- `// Assert`
  - Compare the actual output with the expected output

Test Driven Development (TDD)

> Write tests first, then write code to make the tests pass

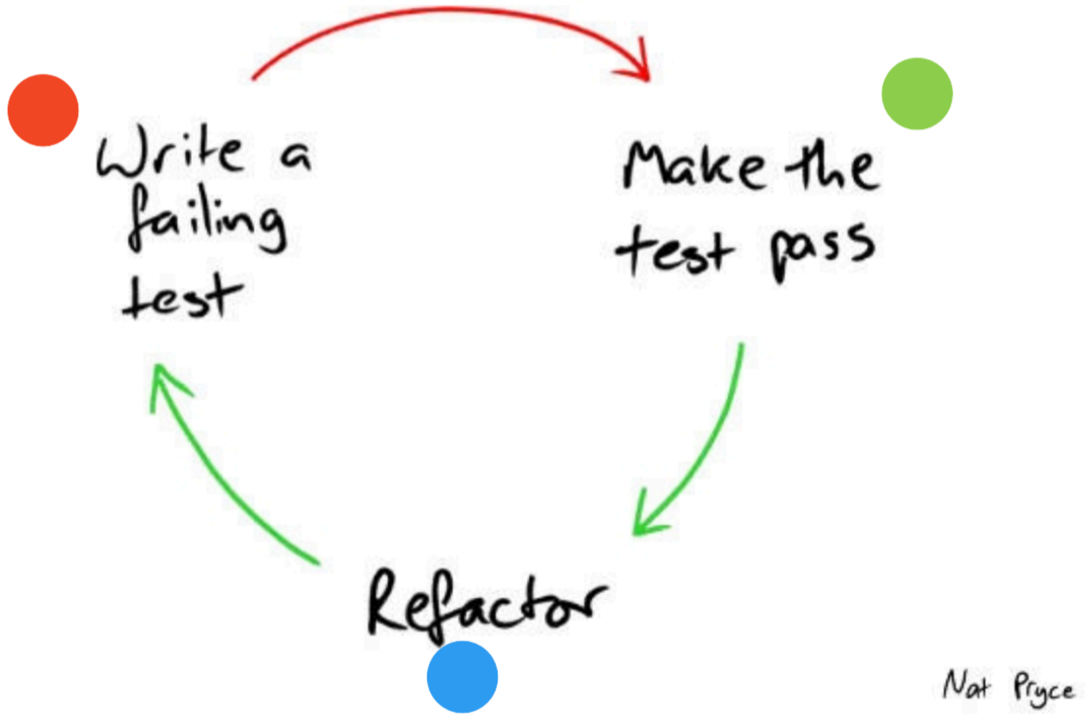
Advantages

- Forces you to think about the *interface* or *design* of your code before writing it
- Leads to more **testable** code which, in turn leads to *generally* higher quality code
- Encourages you to think about:
  - Edge cases
  - Refactoring (variable names, function names, etc.)

Disadvantages

- Requires you to have a lot of context about the problem you’re trying to solve
- Generally very time consuming

How to TDD



What to test (in functions)

- **Inputs:** Think about the different possible inputs
- **Control flow:** Think about the different possible paths through the function
    - `` `if` `` statements
    - Loops
    - Error conditions
- **Outputs:** Think about the different possible outputs

Accounting TDD

Demo