

Is TDD Still Relevant?

Yes, but don't be dumb about it.

Brian Okken

- Lead Software Engineer at Rohde & Schwarz
- Wireless Communications
- Mostly measurements
- C++ : Mostly embedded code
- Python : Testing and automation
- pytest : of course

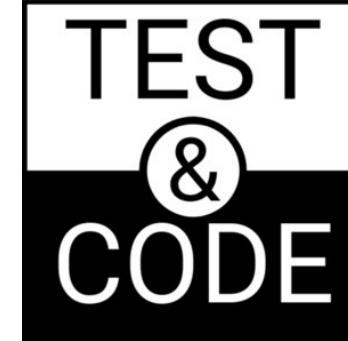
Brian Okken

But you may know me from

- podcasts
- books
- or maybe a pytest course

Podcasts

- Test and Code, 2015 - 2025
- Python Bytes, 2016 - 2026
- Python People, 2023 - 2024

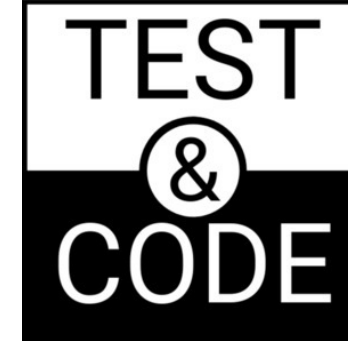


PYTHON BYTES
Python headlines delivered directly to your earbuds



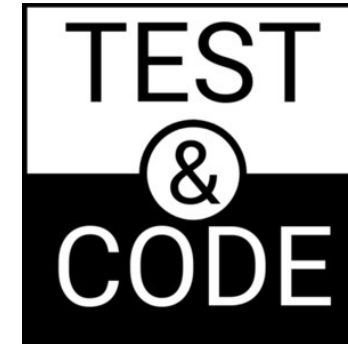
Podcasts

- Test and Code, 2015 - 2025
 - 10 years
 - 238 episodes
- Python Bytes, 2016 - 2026
- Python People, 2023 - 2024



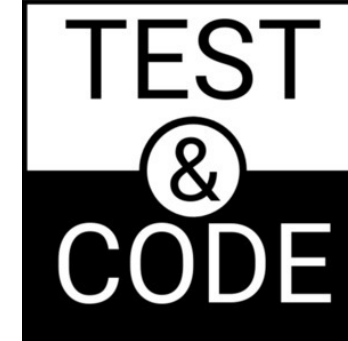
Podcasts

- Test and Code, 2015 - 2025
- Python Bytes, 2016 - 2026
 - almost 10 years
 - 482 episodes when I left
 - almost 500 now
- Python People, 2023 - 2024



Podcasts

- Test and Code, 2015 - 2025
- Python Bytes, 2016 - 2026
- Python People, 2023 - 2024
 - < 1 year
 - 14 episodes



A large, stylized Python logo watermark is centered in the background. It consists of two interlocking snakes, one blue and one yellow, forming a circular shape. The background is a light greenish-blue gradient with faint, repeating text 'Python' in a light blue color.

Python People will Return

<https://pythonpeople.pythontest.com>

14 Amazing Previous Guests

Michael Kennedy
Paul Everitt
Brett Cannon
Barry Warsaw
Bob Belderbos

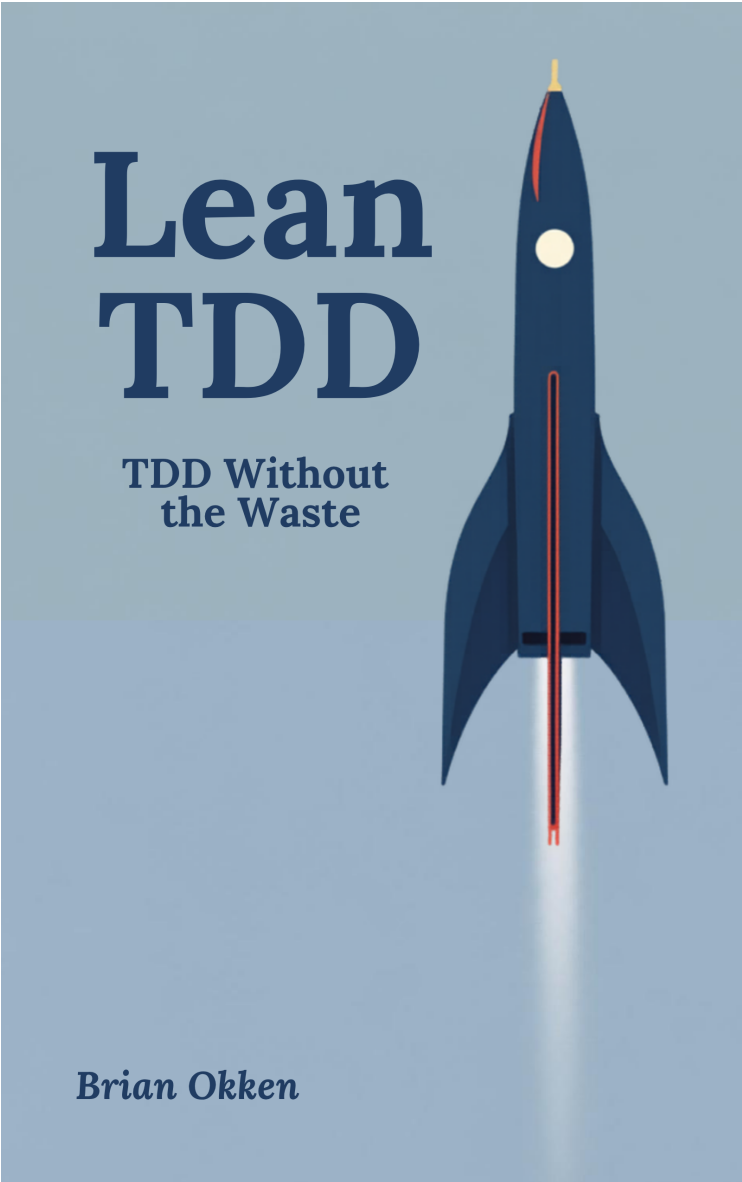
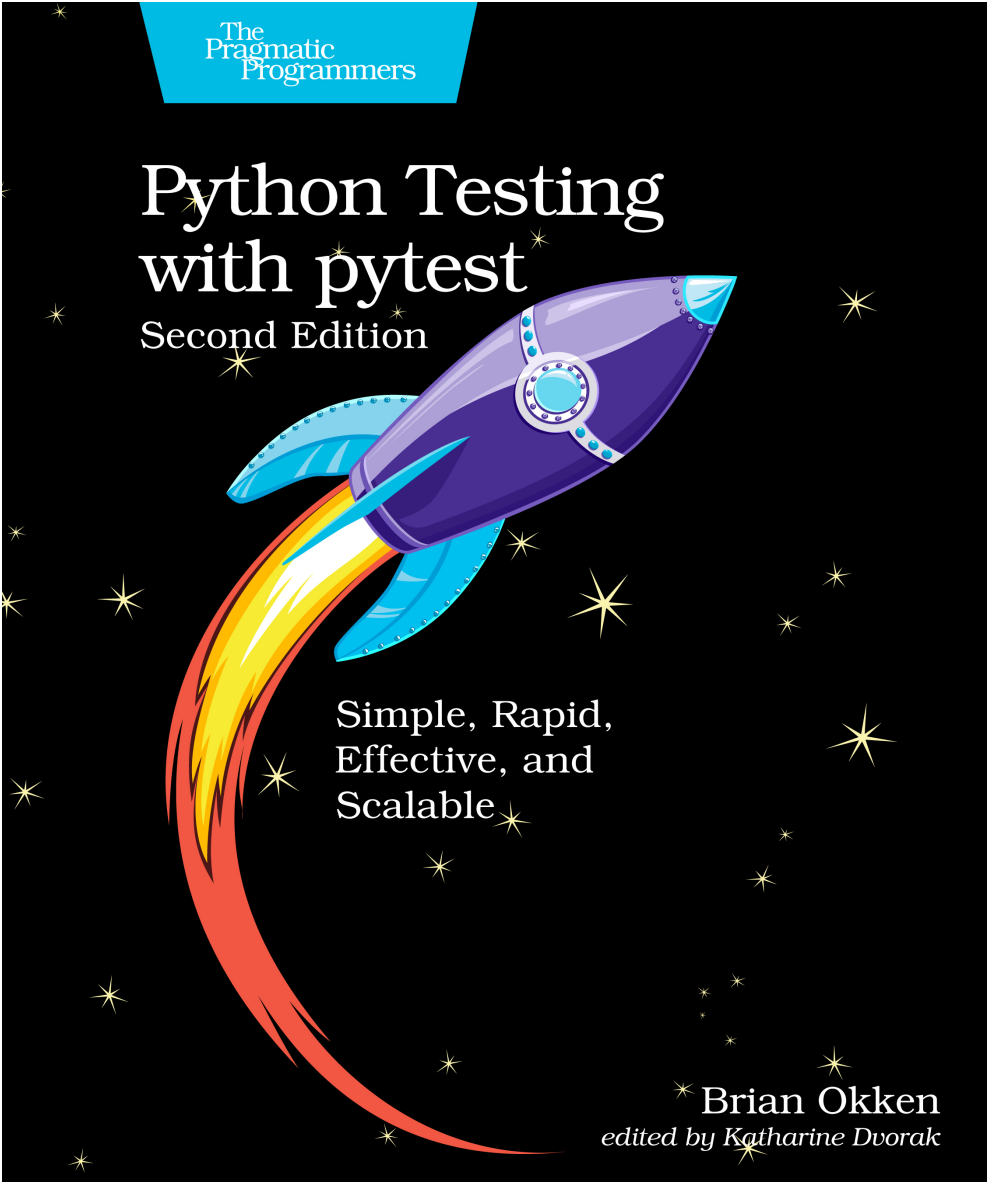
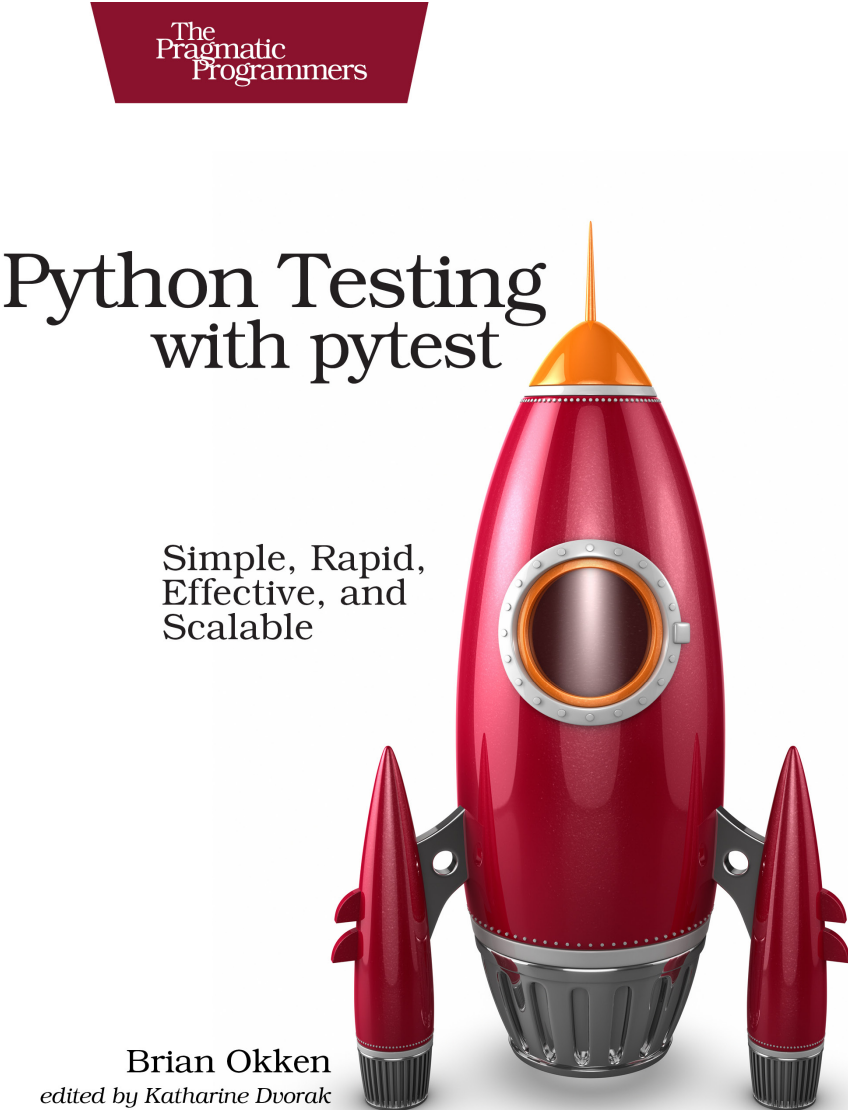
Naomi Ceder
Mariatta Wijaya
Carlton Gibson
Will Vincent
Julian Sequeira

Pamela Fox
Nikita Karamov
Rob Ludwick
Shauna Gordon-McKeon

Future Guests

- I've got a list started
- Let me know if you wanna be on it
- Format might change
- Logo might change
- The people will always be awesome

Books



Courses



Courses



Lead Software Engineer

TDD

My career

A CS nerd in an EE world

My career

Building software (and hardware), with tests

Satellite Test Systems



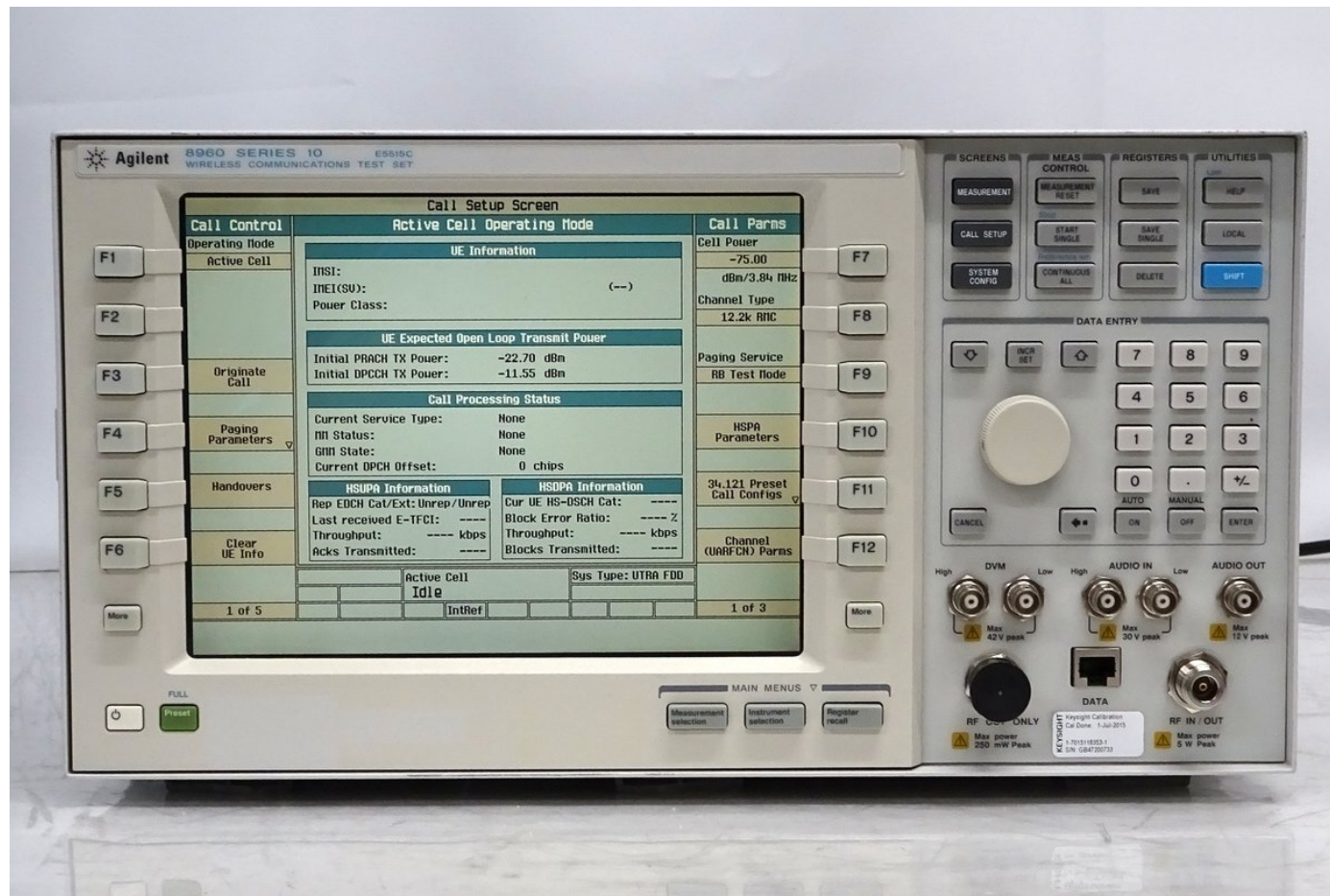
- Started at Hewlett-Packard
- Instrument drivers, GUIs, utilities, ...
- Dev team testing: Yay!
- Manual testing: Boo!

Spectrum Analyzer - Cellular Measurements

- Cell Tower Testing
- Measurements from UI to DSP
- Tests done by a QA team
- QA not embedded, but close



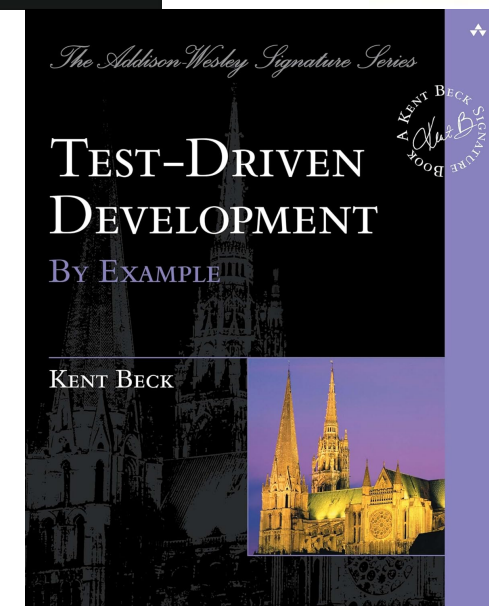
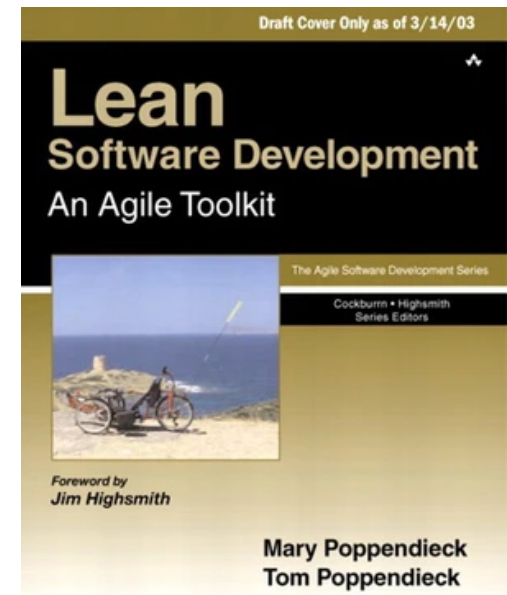
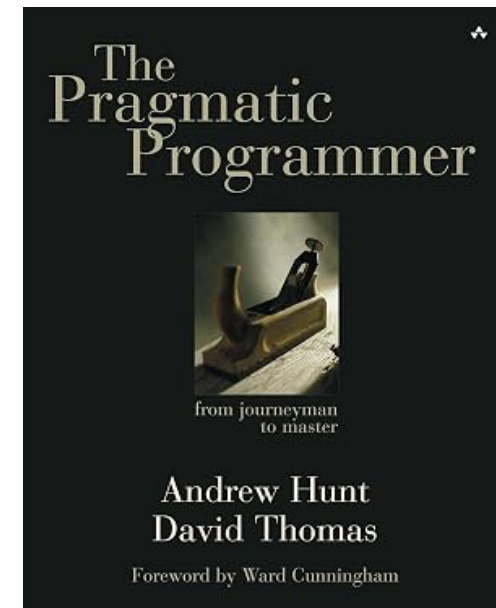
Wireless Communication Tester



- A cell tower in a box
- Hardware isolation
- Separate QA
- Hard to test my code
- So I looked for ways to improve the processes

Process Learning

- Pragmatic Programmer
- Lean Software Development
- Articles on XP & TDD
- TDD by Example



Started trying TDD

TDD with embedded code has challenges

- Compile, load, reboot cycle
- Testing in micro steps is too slow
- Bigger steps was doable
- Needed a debug API to reach my code

Started noticing the weird split
between QA and dev regarding
testing.

Using the same tools, but not working together

- A common Python test framework
- Testing against the public remote interface

But...

- Can't run each others tests

QA

- Test racks with extra equipment
- Final acceptance tests
- Happy path, error cases, etc.
- ALAC or workflow tests
- run regularly

Developer tests

- Individual boxes, maybe two.
- Tests to convince myself my code works
- Mostly happy path
- a.k.a. CYA tests
- not run regularly

After just reading
Lean Software Development
this wastefulness was frustrating

7 kinds of waste

1. Partially done work

2. Extra Processes

3. Extra Features

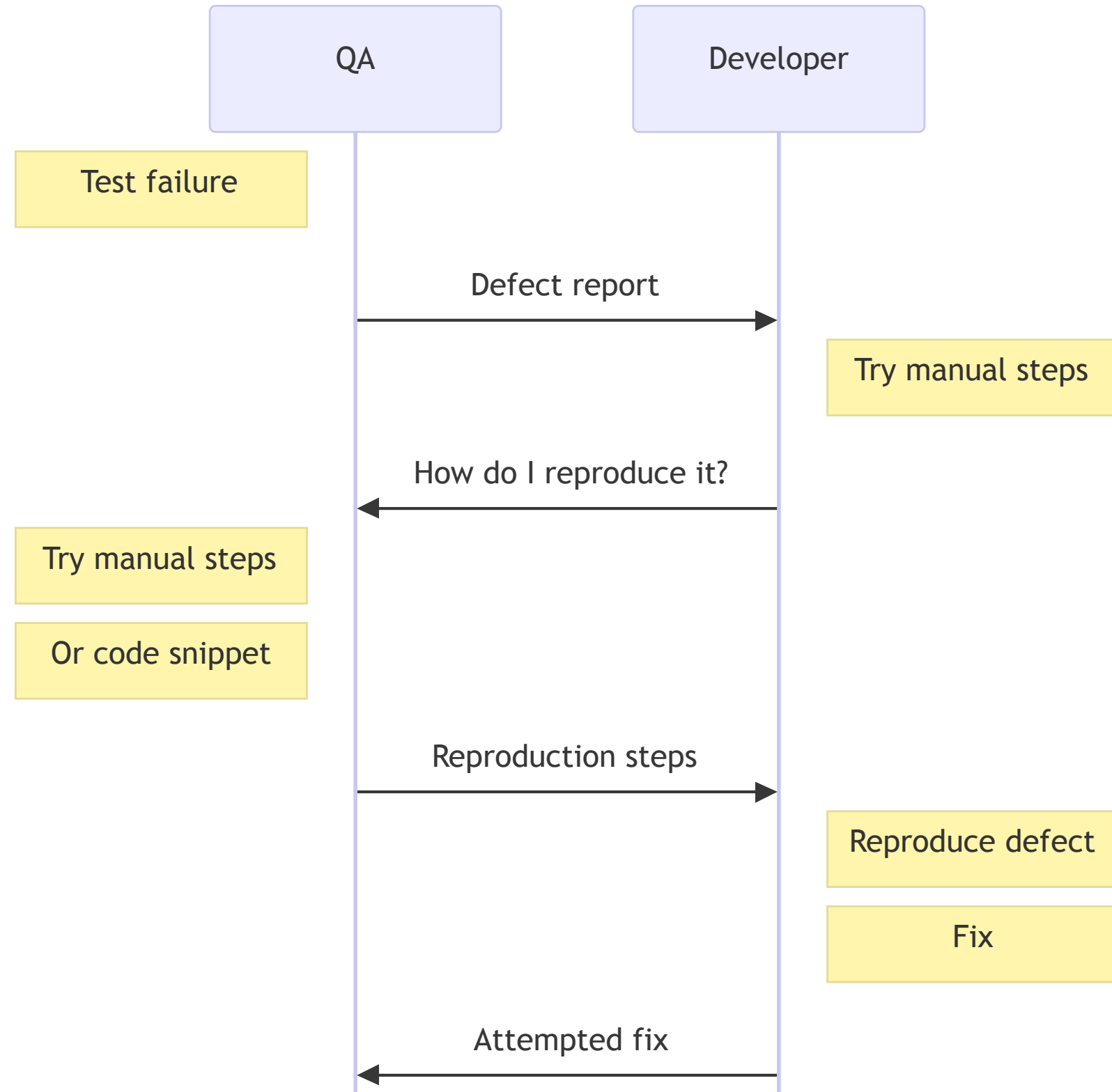
4. Task Switching

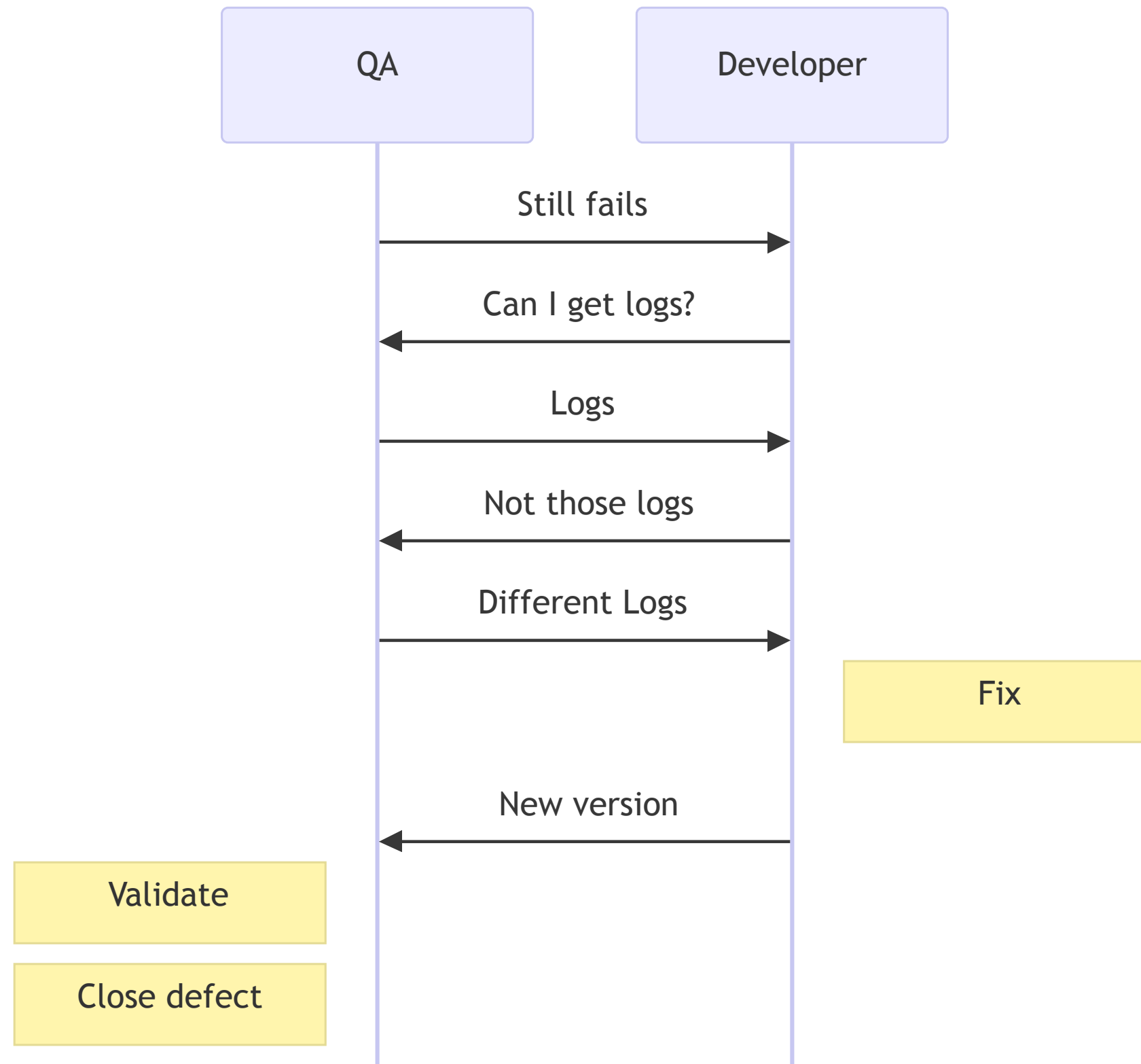
5. Waiting

6. Motion (hand offs)

7. Defects

Defect Ping-Pong

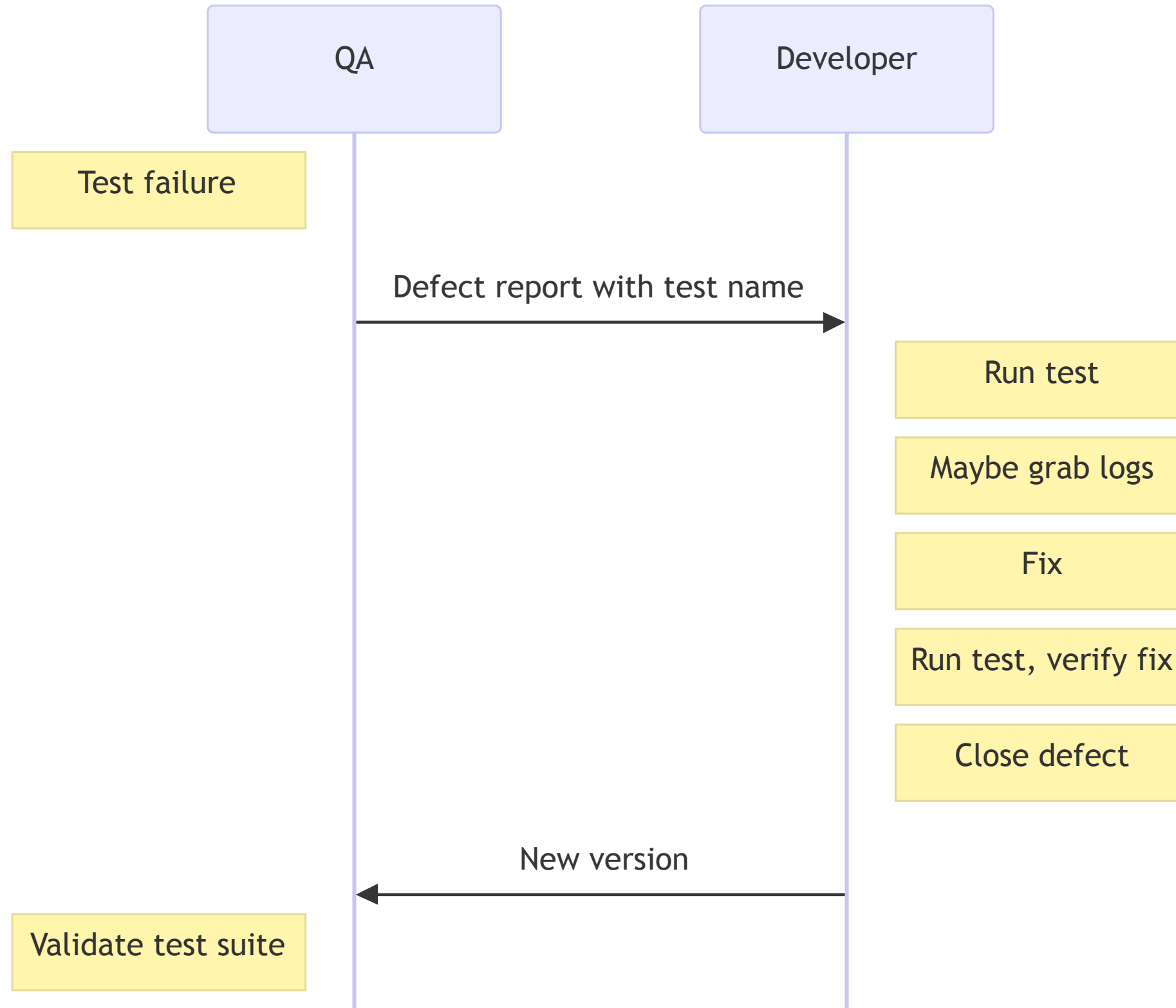




At least of 6 the 7 forms of waste

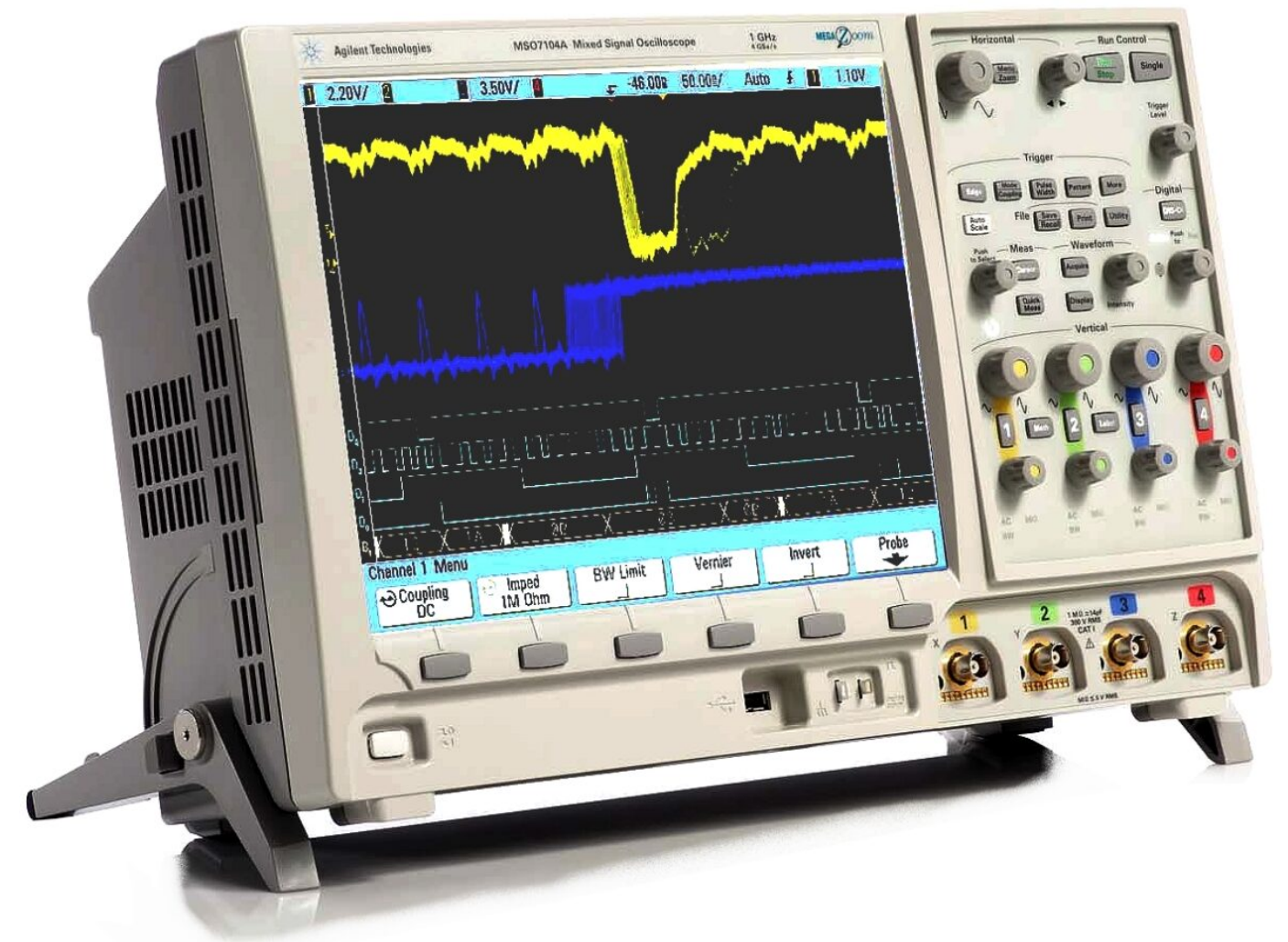
- 1. Partially done work**
- 2. Extra Processes**
3. Extra Features
- 4. Task Switching**
- 5. Waiting**
- 6. Motion (hand offs)**
- 7. Defects**

What it should be



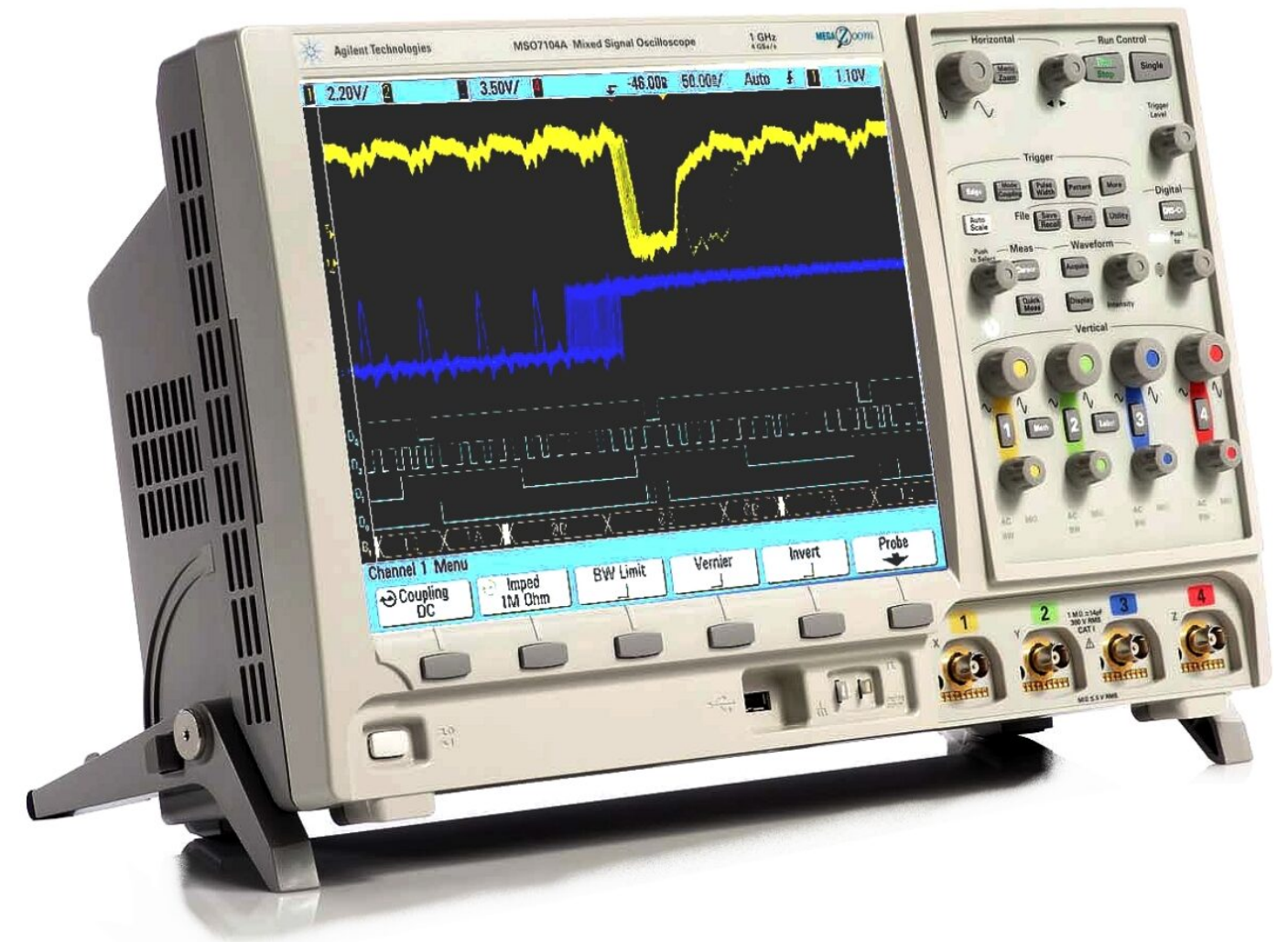
Oscilloscopes

- A lot of manual testing
- Way too much
- I started doing test automation
- And teaching colleagues



Test Automation

- API level tests
- By developers
- Validating features during development
- Development test suite doubles as the acceptance suite



Wireless Communication Tester at R&S



- Test runner was meh
- So I wrote a new one
- Python, of course
- Quickly migrated to pytest

Wireless Communication Tester at R&S

- A team lead for about 10 years
- Tests written by developers and test engineers
- All through the API
- Very few quality issues when dev and test together



A benefit from testing early

Ambiguity is quick to remedy when domain experts are still working on the problem.

Another

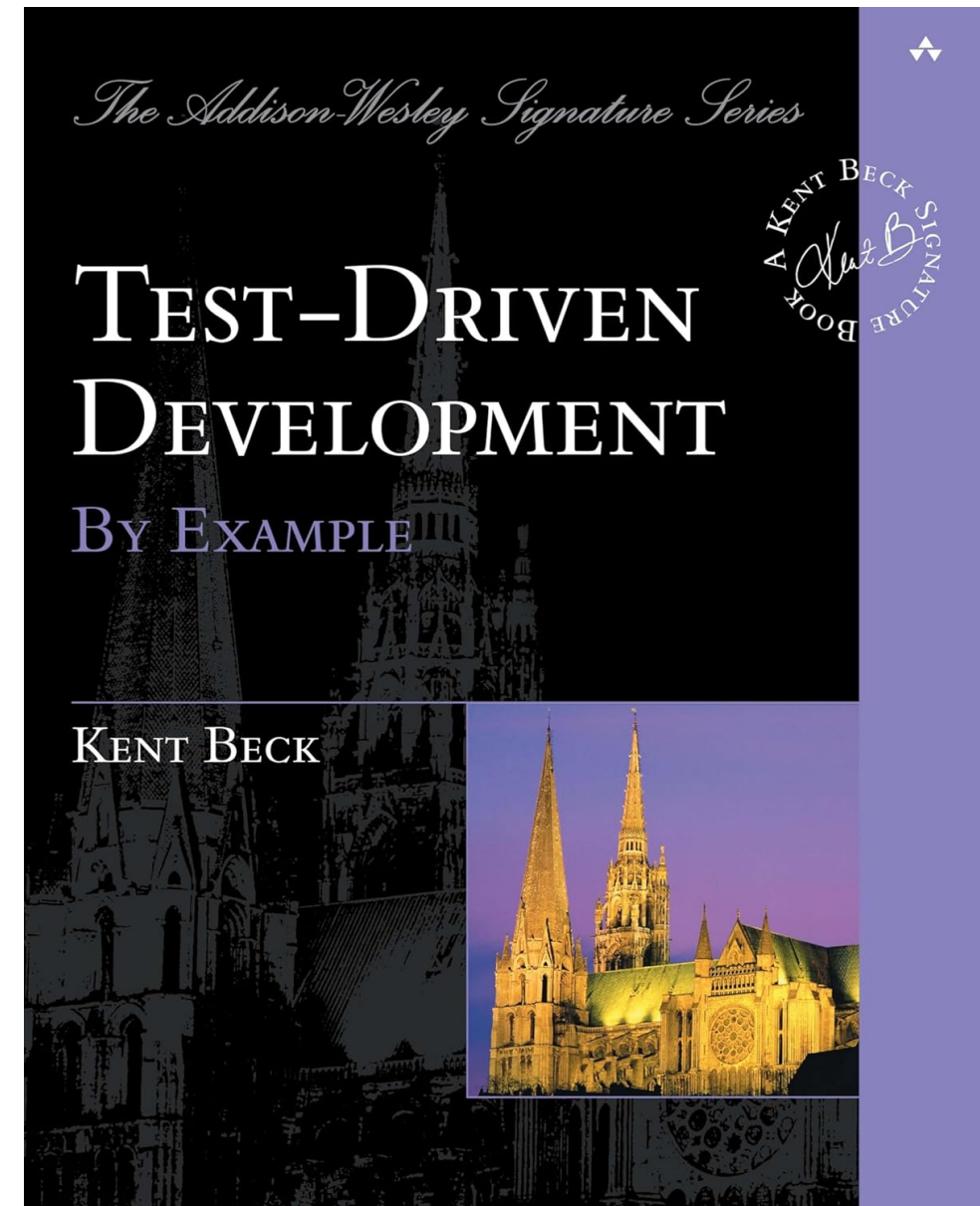
Most problems don't make it to a wide audience

So we should probably get on with it and actually cover ...

TDD : Test Driven Development

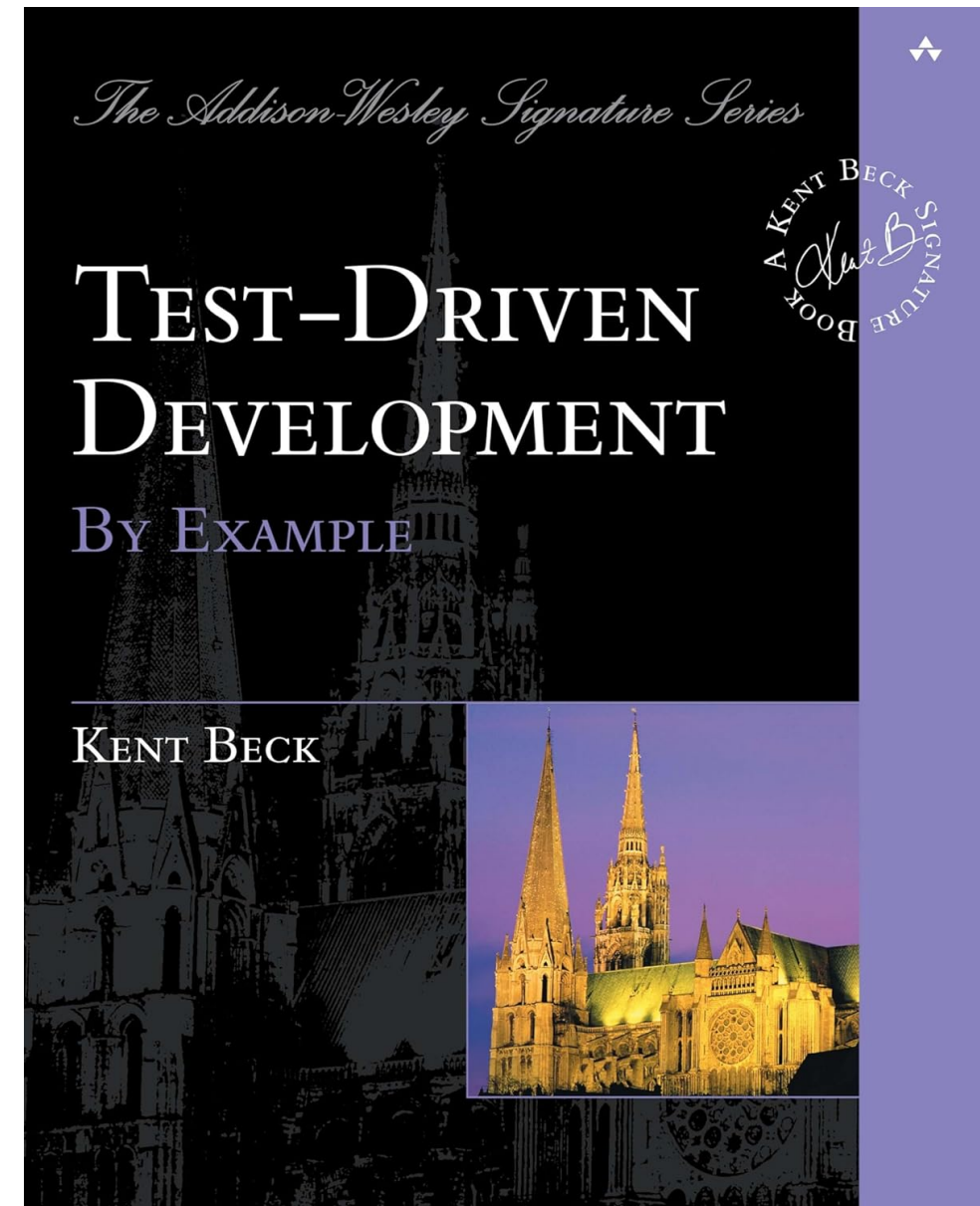
TDD: Test Driven Development

- The Kent Beck one
- Classic TDD
- Classical TDD
- but not TDD Classic
- Too much like ~~Coke Classic~~



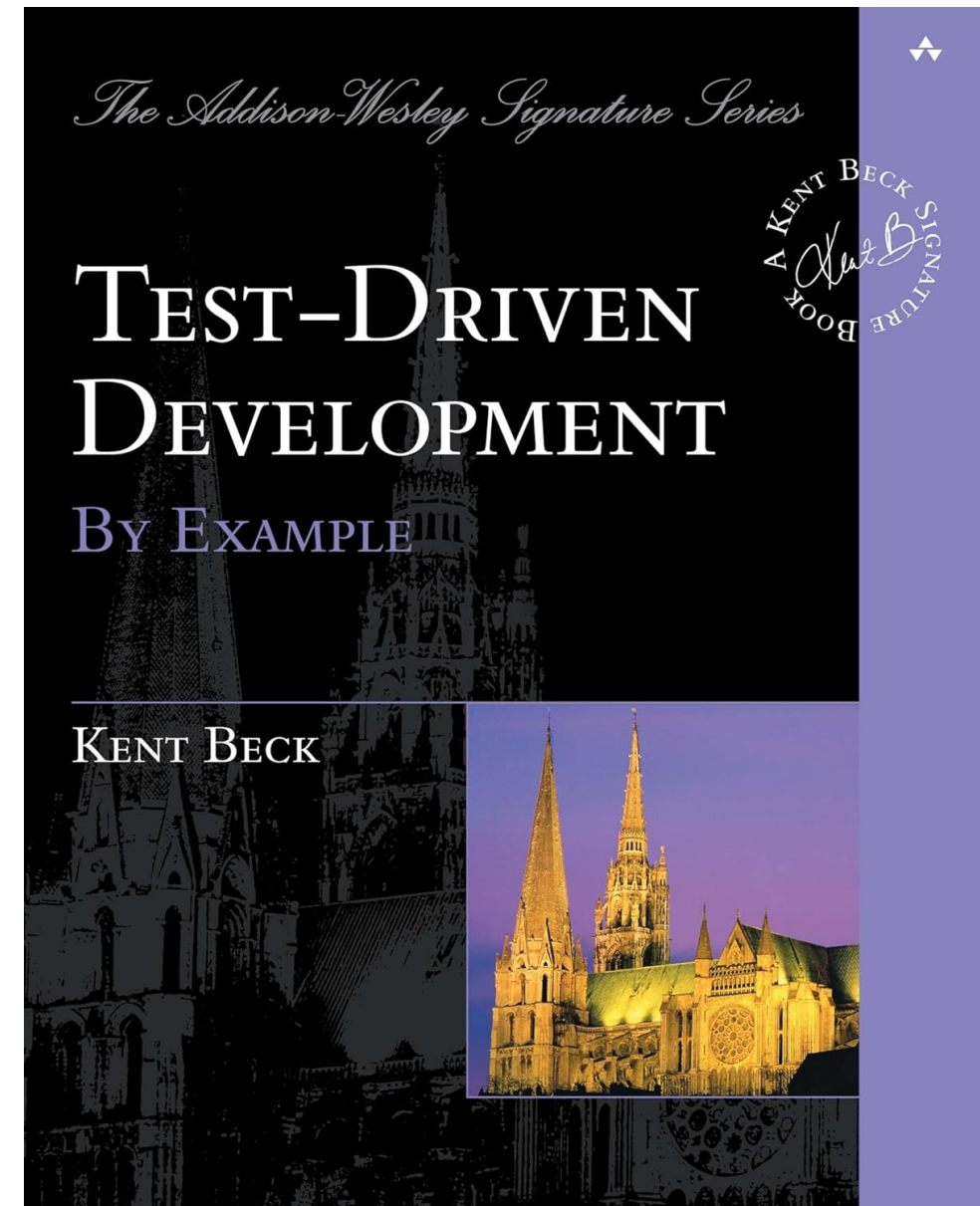
TDD Steps

1. Quickly add a test.
2. Run all tests and see the new one fail.
3. Make a little change.
4. Run all the tests and see them all succeed.
5. Refactor to remove



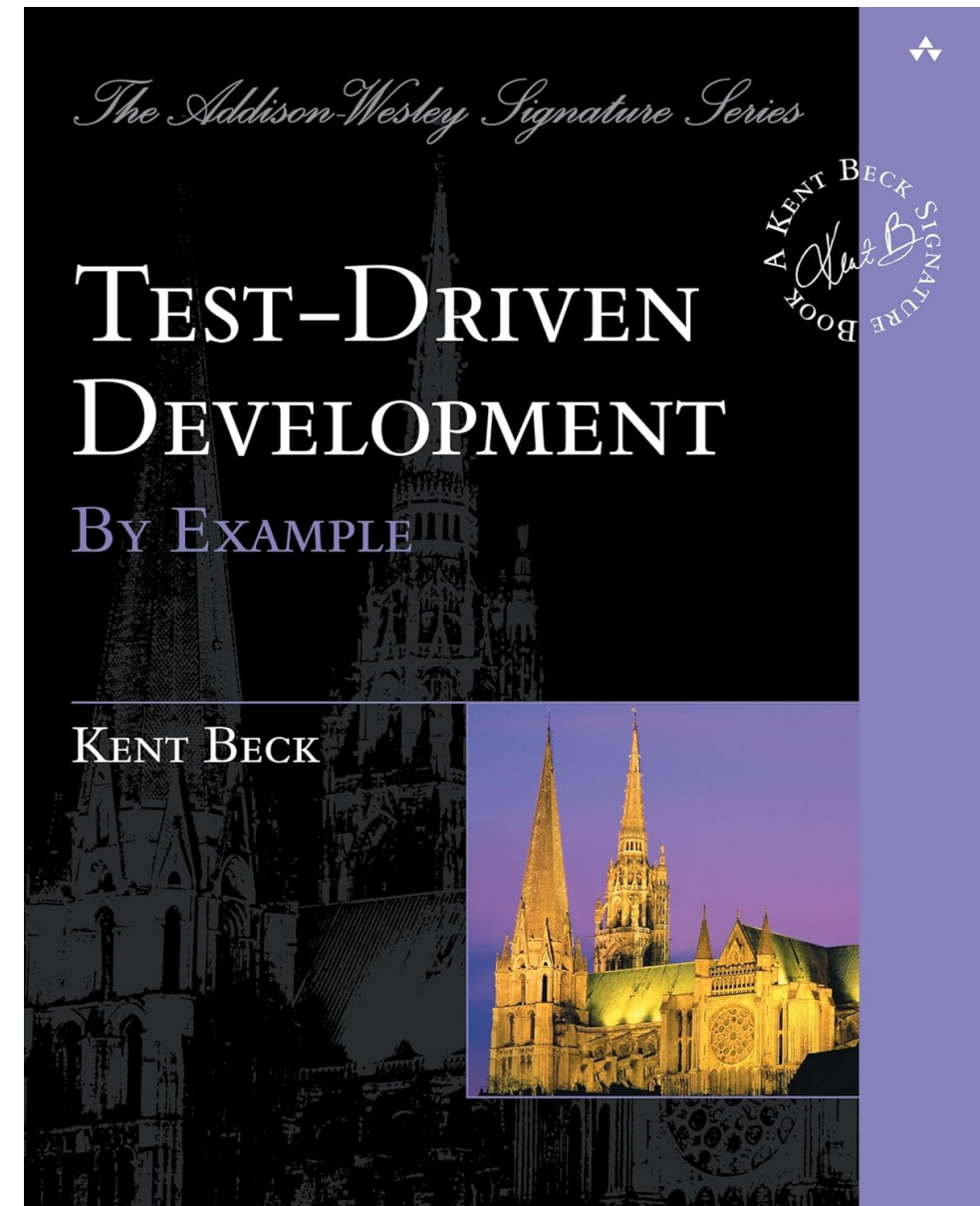
Shorter Version

1. Red
2. Green
3. Refactor



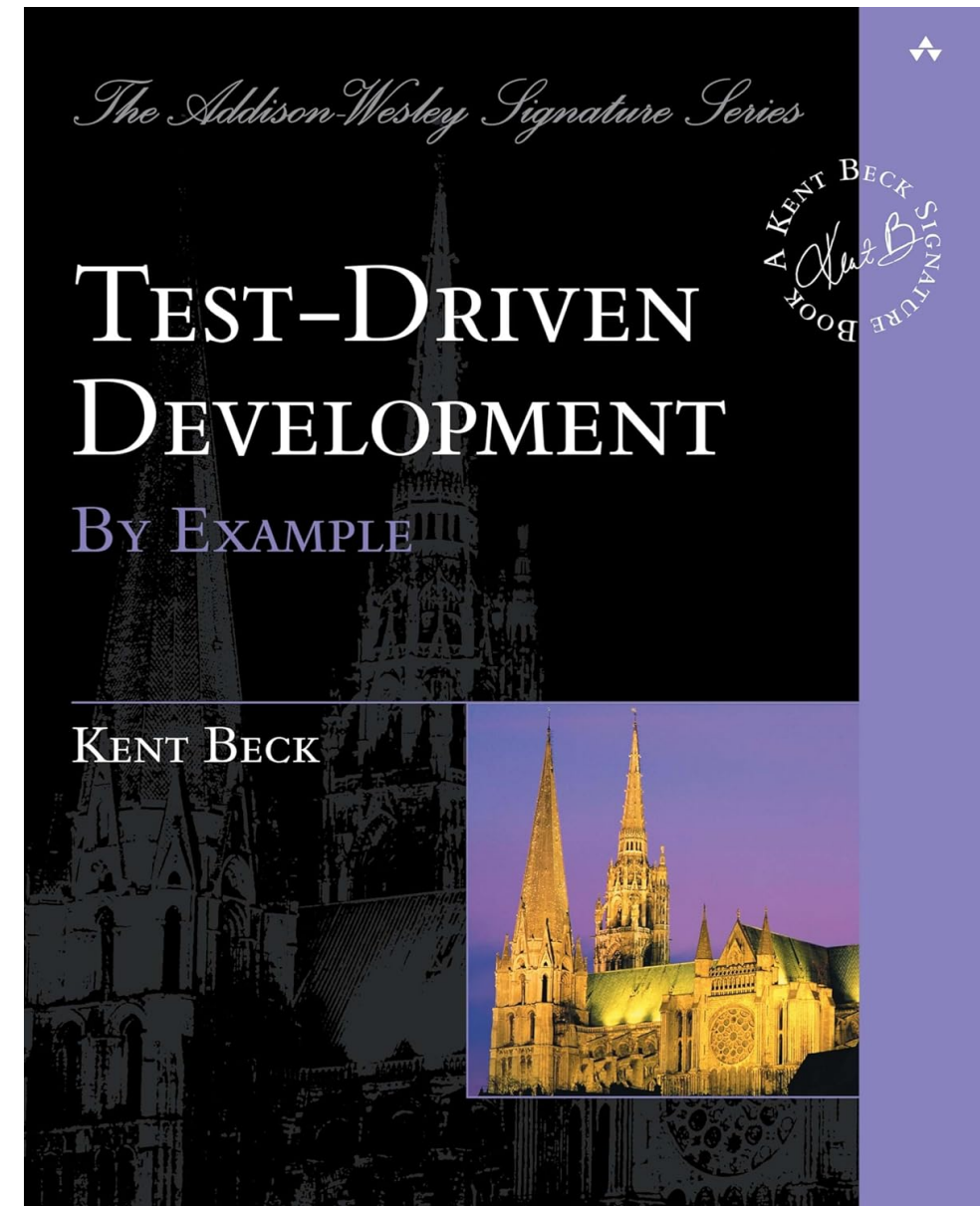
Shorter Version

1. Red - write a failing test
2. Green
3. Refactor



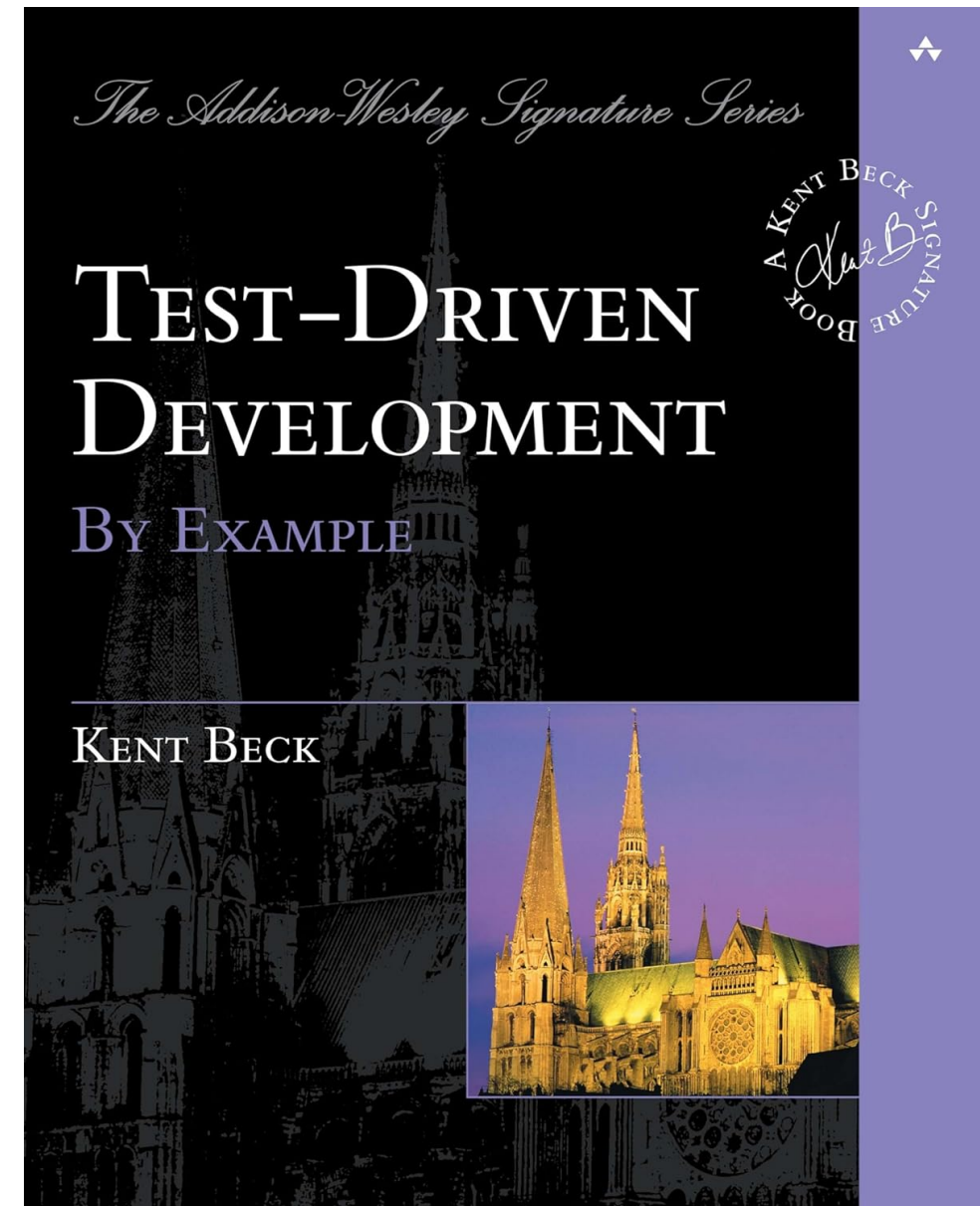
Shorter Version

1. Red - write a failing test
2. Green - write the simplest code to make the test pass
3. Refactor



Shorter Version

1. Red - write a failing test
2. Green - write the simplest code to make the test pass
3. Refactor - clean it up



Questions

- What does "simplest code" mean?
- How do I know what test to write next?
- When am I actually done?

TDD

1. Red - write a failing test
2. Green - write the simplest code to make the test pass
3. Refactor - clean it up

Also, there's a missing piece

The list of test cases

- Kent used it in the book
- Started with a list of things to test
- Added to the list during development
- But most people forget about this list
- It's not in Red/Green/Refactor or even the longer summary

21 Years of Chaos

- TDD factions
- Classic vs Mockist
- Statements like
 - "TDD isn't about testing, it's about design"
- BDD, ATDD, Double loop TDD
- Gherkin

Canon TDD

- 21 years later, in 2023, a blog post from Kent
- To clarify what he meant by TDD

Canon TDD Steps

1. Write a list of test cases you think you need.
2. Pick one, write a test function for it.
3. Write/modify code until the whole suite, including the new test, passes.
4. Optionally refactor.
5. Add any newly-discovered test cases to the list.

Canon TDD Steps - the list is back

1. **Write a list of test cases you think you need.**
2. Pick one, write a test function for it.
3. Write/modify code until the whole suite, including the new test, passes.
4. Optionally refactor.
5. **Add any newly-discovered test cases to the list.**

Canon TDD Steps - refactoring is optional

1. Write a list of test cases you think you need.
2. Pick one, write a test function for it.
3. Write/modify code until the whole suite, including the new test, passes.
4. **Optionally refactor.**
5. Add any newly-discovered test cases to the list.

Canon TDD Steps - no longer simple

1. Write a list of test cases you think you need.
2. Pick one, write a test function for it.
3. Write/modify code until the whole suite, including the new test, passes. **No mention of "simplest code"**
4. Optionally refactor.
5. Add any newly-discovered test cases to the list.

Some good came from the 21 years of chaos

There are lessons in the alternative forms of TDD

Some of the ideas are worth keeping

London / Mockist TDD

- Test against the API. Yay!
- Tons of mocks. Boo!
- Tests implementation
- Hard to read

BDD: Behavior Driven Development

- Behaviors: Yay!
- Given/When/Then: Yay!
- Domain Specific Language DSL: Yay!
- Gherkin: Boo!

ATDD: Acceptance Test Driven Development

- Similar to BDD
- A team workflow. Yay!
- Focus on acceptance tests: Yay!

Lets take all the good stuff

Leave out the waste

and create ...

Lean TDD

Lean TDD

- Take Canon TDD
- Test against the API - *from London TDD*
- Test behavior, not implementation - *from BDD and Canon TDD*
- Start with acceptance criteria - *from ATDD*
- Scale process up and down as needed - *using Lean*

Lean TDD

Steps and Strategies

Lean TDD Steps

1. List the acceptance criteria
2. Expand one into test cases
3. Write a test
4. Make it pass
5. Optionally refactor
6. Add to lists as needed
7. Repeat until done

Lean TDD Steps

1. **List the acceptance criteria**
2. Expand one into test cases
3. Write a test
4. Make it pass
5. Optionally refactor
6. Add to **lists** as needed
7. Repeat until done

Canon TDD Steps

1. List of test cases
2. Write a test
3. Make it pass
4. Optionally refactor.
5. Add to **list** as needed
6. Repeat until done

Lean TDD Steps

1. List the acceptance criteria
2. Expand one into test cases
3. Write a test
4. Make it pass
5. Optionally refactor
6. Add to lists as needed
7. Repeat until done

But I need a reminder to run the
whole test suite

Lean TDD Steps - Full list

1. List acceptance criteria
2. **Run the test suite**
3. Expand one into test cases
4. Write a test
5. Make it pass
6. **Run the test suite**
7. Optionally refactor
8. Add to lists as needed
9. Repeat until done

Lean TDD Strategies

- Book also includes 13 strategies

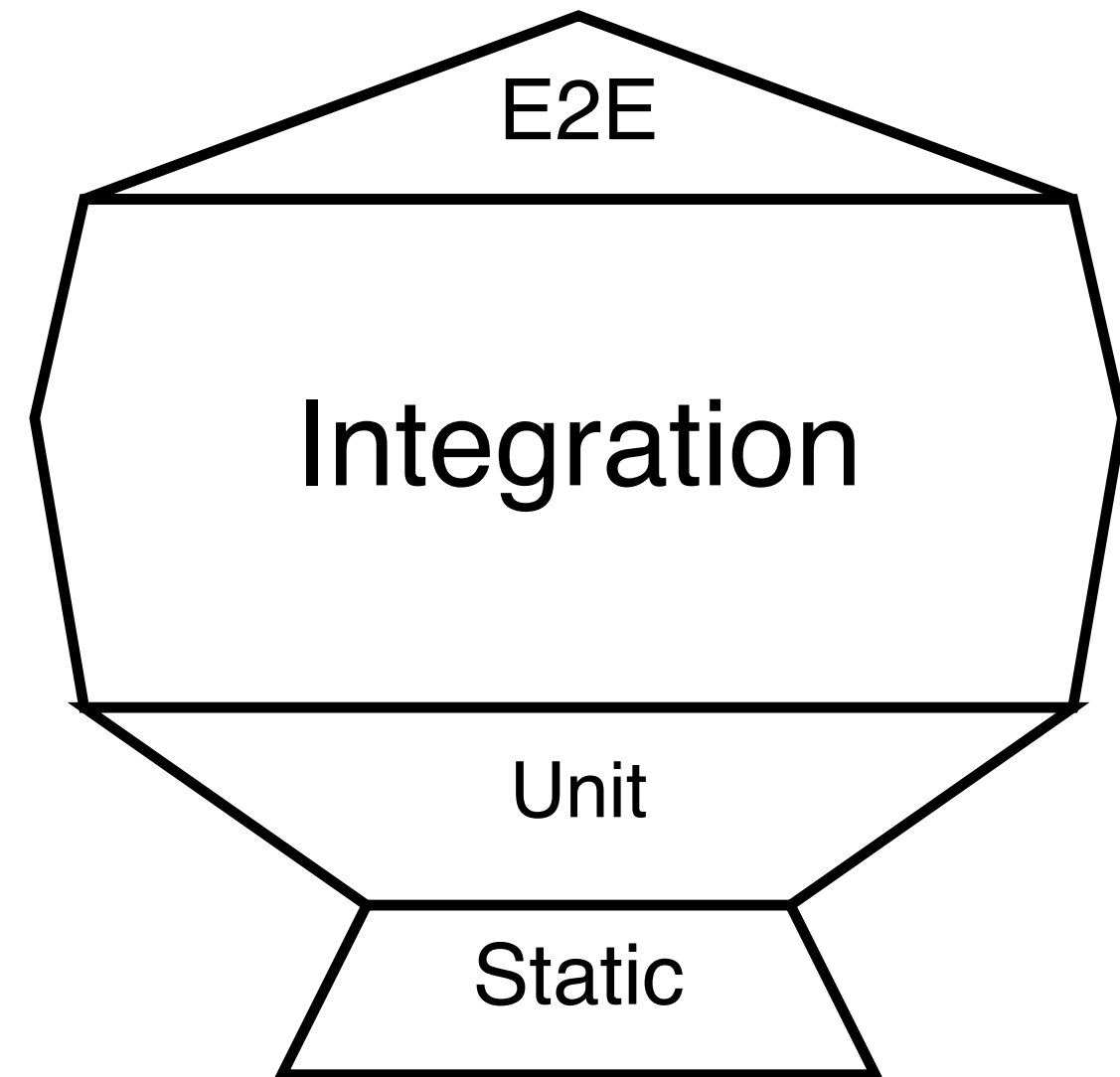
A few strategies

- Test primarily through a system API
- Design for an internal system API if necessary
- Add lower level testing as needed
- Minimize tests tightly coupled to implementation
- Use static analysis
- Use the testing trophy

The Testing Trophy

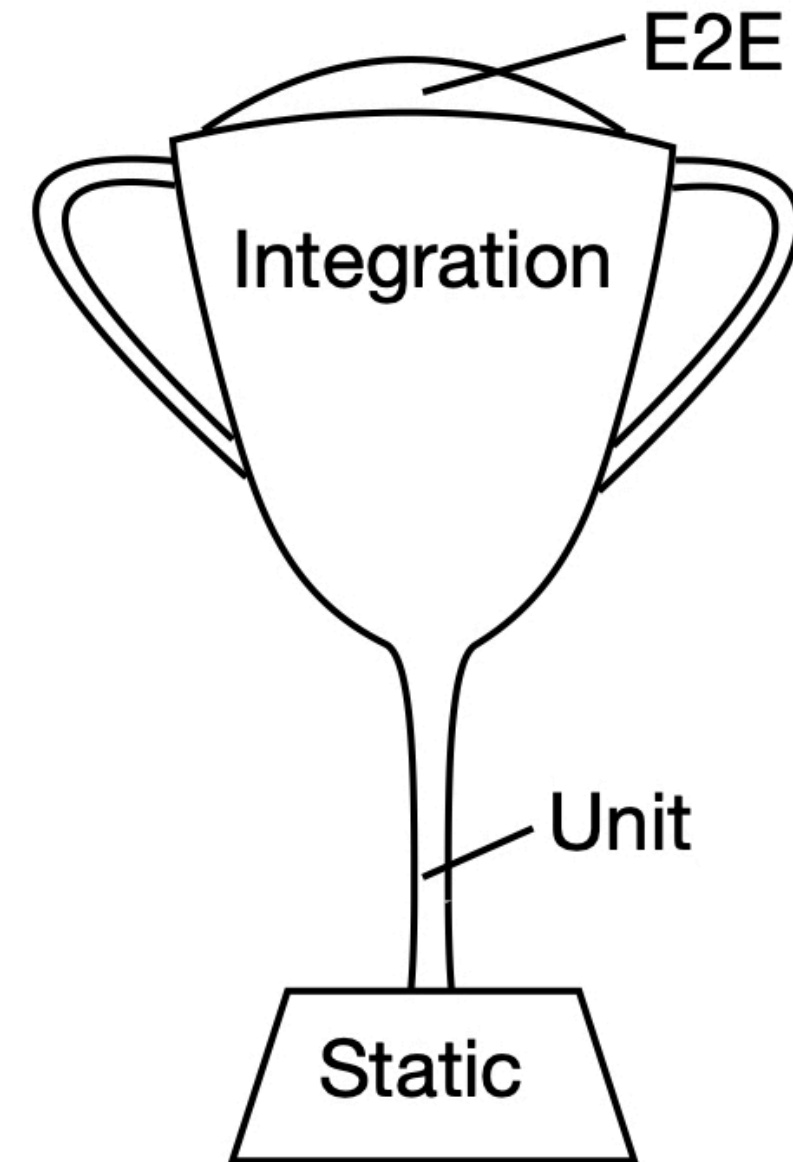
Testing Trophy

- Kent C Dodds
- "Write tests. Not too many. Mostly integration."
- A blog post by Dodds
- based on a tweet from Guillermo Rauch



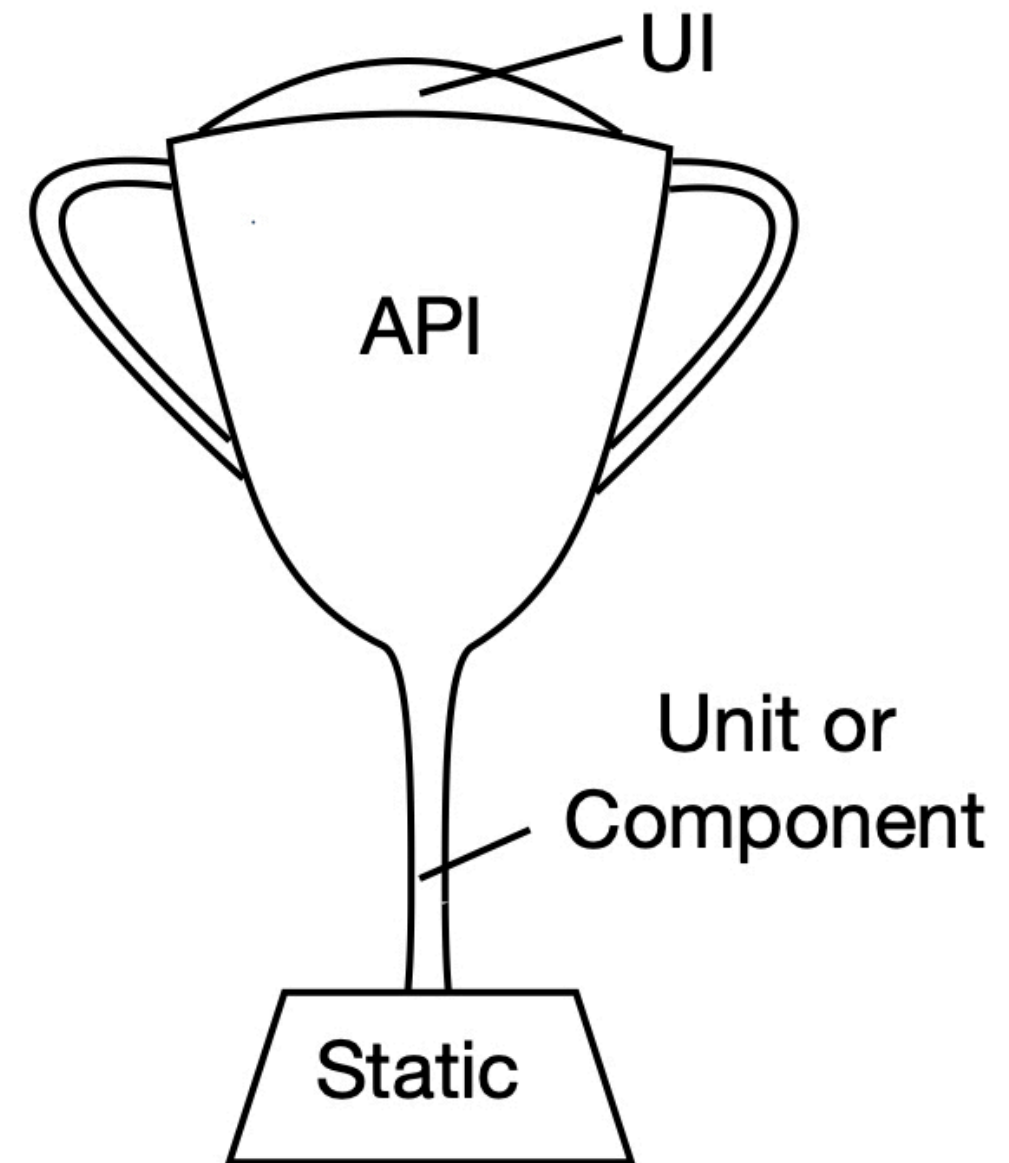
Testing Trophy - my drawing

- My attempt at drawing a trophy
- The labels might make sense for js
- Don't really match terms I'm used to when talking about testing.



Testing Trophy - my version

- Don't do many UI tests
- Mostly test through the API
- As high up as is reasonable
- Not too many unit and or component tests
- Be sure to use static analysis



What about AI?

Does this work with AI?

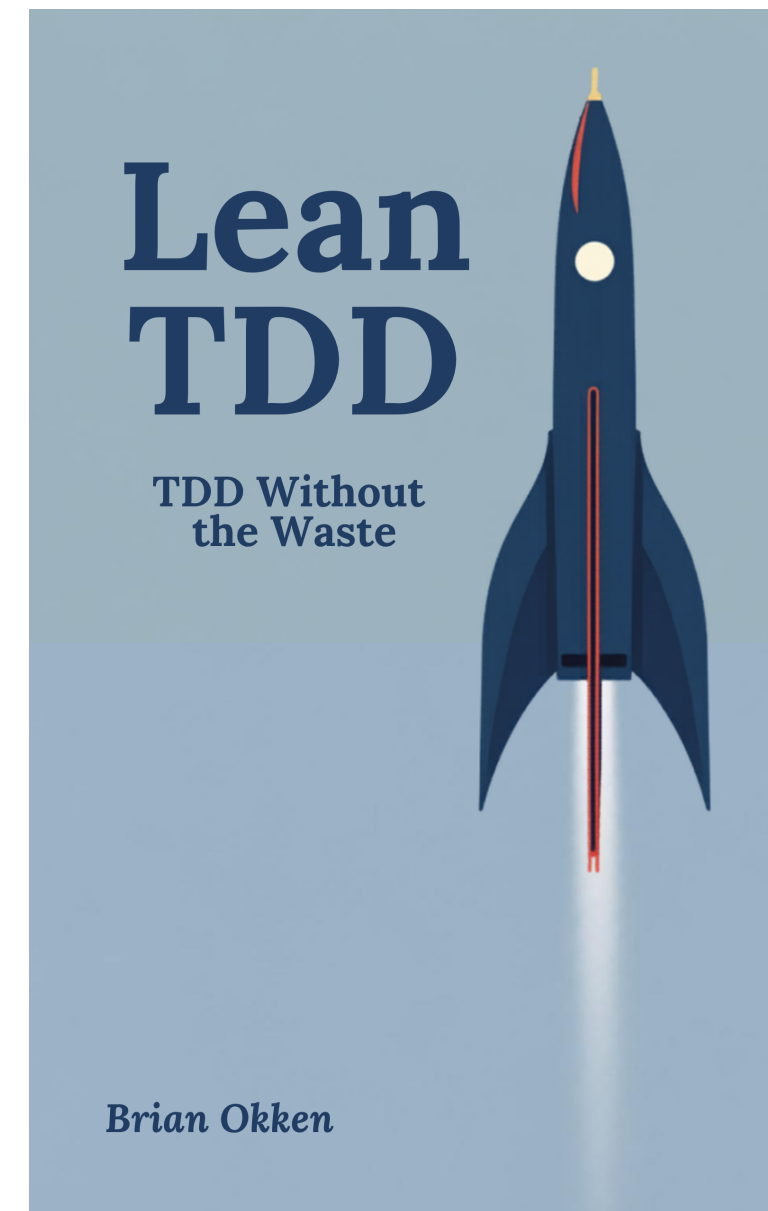
- It's actually a great fit

Coding and Testing with AI

- We care even more about system behavior now
- And less about implementation details
- So higher level tests are more valuable

Way more detail in the book

- Landing page: leantdd.com
- Text version is ~100 pages
- Audio book is about 2 hours (or less depending on speed)
- Non-Amazon versions will be start to become available in November



Thank You

- leantdd.com
Lean TDD book
paperback, digital, and audio versions
- pythontest.com
training, courses, books, blog
- [@brianokken@fosstodon.org](https://fosstodon.org/@brianokken)
Mastodon
- [@brianokken@fosstodon.org](https://fosstodon.org/@brianokken)
Bluesky

