Test-Driven Development (TDD)

Chung-Kil Hur

(Credit: Byung-Gon Chun & from UCB CS169
taught by Armando Fox, David Patterson)

SWPP, CSE, SNU

Coverage, Unit vs. Integration Tests

How much testing is enough?

- Bad: “Until time to ship”
- A bit better: (Lines of test) / (Lines of code)
  - 1.2–1.5 not unreasonable
  - often *much higher* for production systems
- Better question: “How thorough is my testing?”
  - Formal methods
  - Coverage measurement
  - We focus on the latter, though the former is gaining steady traction

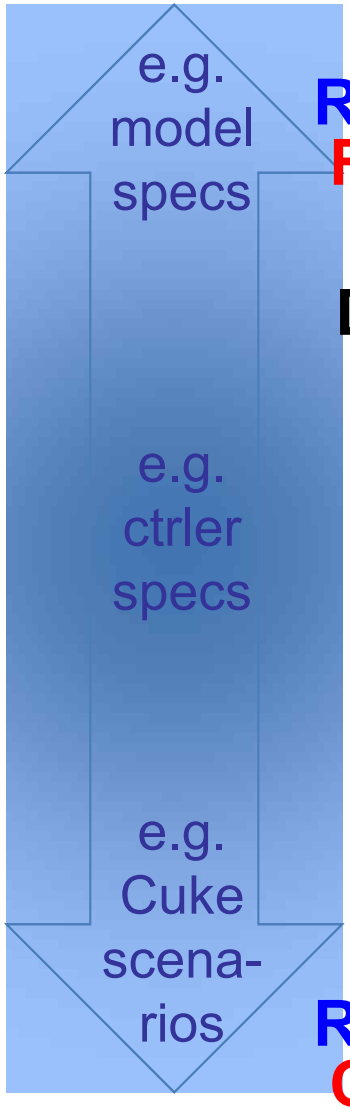
Measuring Coverage—Basics

```
class MyClass
 def foo(x,y,z)
  if x
    if (y && z) then bar(0) end
   else
    bar(1)
   end
 end
 def bar(x) ; @w = x ; end
end
```

- S0: every method called
- S1: every method *from every call site*
- C0: every statement
  - Ruby SimpleCov gem
- C1: every branch in both directions
- C1+decision coverage: every *subexpression* in conditional
- C2: every path (difficult, and disagreement on how valuable)

What kinds of tests?

- Unit (one method/class)
- Functional or modu (a few methods/classes)
- Integration/system



Runs fast **High coverage**
Fine resolution

Many mocks;
Doesn't test interfaces

Few mocks;
tests interfaces

Runs slow **Low coverage**
Coarse resolution

Going to extremes

- ✗ “I kicked the tires, it works”
- ✗ “Don’t ship until 100% covered & green”
- ☑ use coverage to identify untested or undertested parts of code
- ✗ “Focus on unit tests, they’re more thorough”
- ✗ “Focus on integration tests, they’re more realistic”
- ☑ each finds bugs the other misses

Which of these is POOR advice for TDD?

- ☐ Mock & stub early & often in unit tests
- ☐ Aim for high unit test coverage
- ☐ Sometimes it's OK to use stubs & mocks in integration tests
- ☐ Unit tests give you higher confidence of system correctness than integration tests

Other Testing Concepts; Testing vs. Debugging

Other testing terms you may hear

- Mutation testing: if introduce deliberate error in code, does some test break?
- Fuzz testing: 10,000 monkeys throw random input at your code
  - Find ~20% MS bugs, crash ~25% Unix utilities
  - *Tests app the way it wasn't meant to be used*
- DU-coverage: is every pair <define x/use x> executed?
- Black-box vs. white-box/glass-box

TDD vs. Conventional debugging

Conventional	TDD
Write 10s of lines, run, hit bug: break out debugger	Write a few lines, with test first; know immediately if broken
Insert printf's to print variables while running repeatedly	Test short pieces of code using expectations
Stop in debugger, tweak/set variables to control code path	Use mocks and stubs to control code path
Dammit, I thought for sure I fixed it, now have to do this all again	Re-run test automatically

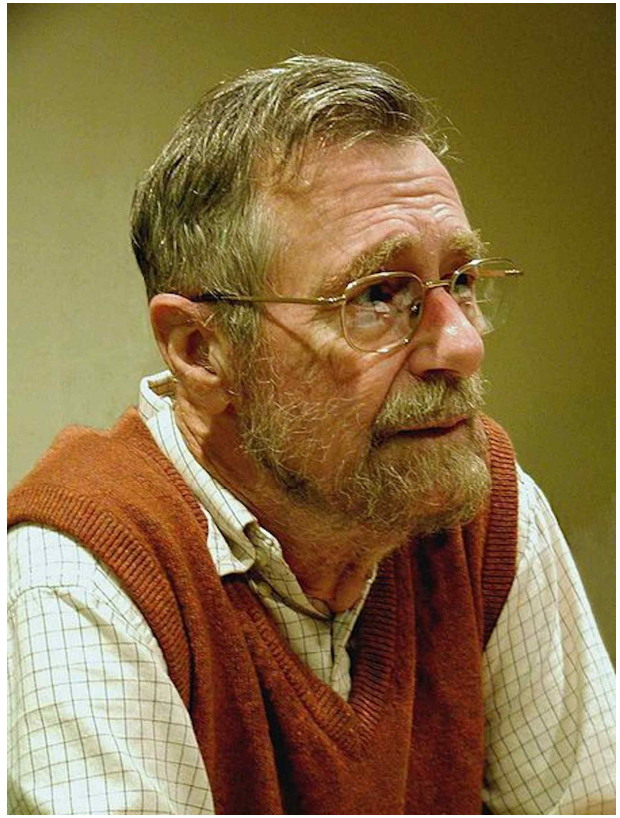
- Lesson 1: TDD uses same skills & techniques as conventional debugging—but more productive (FIRST)

- Lesson 2: writing tests *before* code takes *more time* up-front, but often *less time* overall

Limits of Testing

- Program testing can be used to show the presence of bugs, but never to show their absence!
  - – Edsger W. Dijkstra

(received the 1972 Turing Award for fundamental contributions to developing programming languages)



Formal Methods

- Start with formal specification & prove program behavior follows spec.

1. Human does proof
2. Computer via automatic theorem proving
   - Uses inference + logical axioms to produce proofs from scratch
3. Computer via model checking
   - Verifies selected properties by exhaustive search of all possible states that a system could enter during execution

Formal Methods

- Computationally expensive, so use
  - Small, fixed function
  - Expensive to repair, very hard to test
  - E.g. Network protocols, safety critical SW

- Biggest: OS kernel
  10K LOC @ $500/LOC
  - NASA SW $80/LOC

Plan-And-Document Perspective on Software Testing

P&D Testing?

- BDD/TDD writes tests before code
  - When do P&D developers write tests?
- BDD/TDD starts from user stories
  - Where do P&D developers start?
- BDD/TDD developers write tests & code
  - Does P&D use same or different people for testing and coding?
- What does the Testing Plan and Testing Documentation look like?

SW Testing: P&D vs. Agile

Tasks	*In Plan and Document*	*In Agile*
Test Plan and Documentation	Software Test Documentation such as IEEE Standard 829-2008	User stories
Order of Coding and Testing	1. Code units 2. Unit test 3. Module test 4. Integration test 5. System test 6. Acceptance test	1. Acceptance test 2. Integration test 3. Module test 4. Unit test 5. Code units
Testers	Developers for unit tests; QA testers for module, integration, system, and acceptance tests	Developers
When Testing Stops	Company policy (e.g., statement coverage, happy and sad user inputs)	All tests pass (green)

TDD Summary

- Red – Green – Refactor, and always have working code
- Test *one* behavior at a time, using seams
- Use `it` "placeholders" or `pending` to note tests you know you'll need
- Read & understand coverage reports
- "Defense in depth": don't rely too heavily on any *one* kind of test