Intro to Test-Driven Development

Jon Reid
@qcoding
qualitycoding.org
Adding tests after the code is complete

@qcoding
https://www.flickr.com/photos/gaz/2717701
3 Steps: “TDD Waltz”

1. Test:
   - Make it verifiable
   - Write a failing test

2. Code:
   - Make it work
   - Make the test pass

3. Design:
   - Make it right
   - Refactor
Write a failing test → Make the test pass
3 Laws of TDD

1. You are not allowed to write any production code unless it is to make a failing unit test pass.

2. You are not allowed to write any more of a unit test than is sufficient to fail; and compilation failures are failures.

3. You are not allowed to write any more production code than is sufficient to pass the one failing unit test.
FizzBuzz

• Create a function that will return “Fizz” if the positive integer argument is a multiple of 3.

• The function should return “Buzz” if the argument is a multiple of 5.

• If the argument is a multiple of both 3 and 5, it should return “FizzBuzz”.

• Otherwise, it should return the argument in string form.
Ground Rules for Code Demo

“No shame” zone 😞

Yield to the less privileged voice 🙊
What Things Struck You?

1. Write a failing test
2. Refactor
3. Make the test pass

3 Steps — 3 Laws
Barriers & Obstacles 🚫

...What’s stopping you?

Management barriers
- Takes longer
- No firm requirements

Technical barriers
- Large data sets may mean slow or super-complicated testing
- Lack of tools
- Legacy code
- Asynchronous events: How do I even test this thing?
- Service calls -> mocks
- Group buy-in

@qcoding
Realizing quality improvement through test driven development: results and experiences of four industrial teams

Nachiappan Nagappan, E. Michael Maximilien, Thirumalesh Bhat, Laurie Williams

Abstract

Test-driven development (TDD) is a software development practice that has been used sporadically for decades. With this practice, a software engineer cycles minute-by-minute between writing failing unit tests and writing implementation code to pass those tests. Test-driven development has recently re-emerged as a critical enabling practice of agile software development methodologies. However, little empirical evidence supports or refutes the utility of this practice in an industrial context. Case studies were conducted with three development teams at Microsoft and one at IBM that have adopted TDD. The results of the case studies indicate that the pre-release defect density of the four products decreased between 40% and 90% relative to similar projects that did not use the TDD practice. Subjectively, the teams experienced a 15–35% increase in initial development time after adopting TDD.
Another interesting observation from the outcome measures in Table 3 is the increase in time to develop the features attributed to the usage of the TDD practice, as subjectively estimated by management. The increase in development time ranges from 15% to 35%. From an efficacy perspective this increase in development time is offset by the reduced maintenance costs due to the improvement in quality (Erdogmus and Williams 2003), an observation that was backed up the product teams at Microsoft and IBM.
The code is done!

@qcoding
https://www.flickr.com/photos/soldiersmediacenter/8093399583
Where do you get permission for TDD?

...from within yourself
TDD “Paint by Numbers”

- Clojure
- C++
- C#
- Java
- JavaScript
- Objective-C
- Swift

github.com/jlangr/name-normalizer
TDD Exercises

- Bowling Game
- Roman Numerals
- codingdojo.org/kata
Actual Work in “Safe Zone” 🧵

- Get your build system to run tests & report failures
- Avoid difficult dependencies — learn Hexagonal Architecture from Ted Young
- If build time is too long, break your system into modules
I can code! (It sometimes works)

I can add `print()` to diagnose

I have debugger skills

I don’t need the debugger