Test Driven Development

Charles Carter[1]

September 27, 2017

[1]Adapted from http://people.cs.aau.dk/\protect\unhbox\voidb@x\penalty\@M\{}bnielsen/TOV08/lektioner/tdd.pdf

Table of Contents

Mansel andHusted: JUnit in Action

"Test-driven Development is a programming practice that instructs developers to write new code only if an automated test has failed, and to eliminate duplication. The goal of TDD is clean code that works"

Agile Alliance

"Test Driven Development is the craft of producing automated tests for production code, and using that process to drive design and programming."
"For every bit of functionality, you first develop a test that specifies and validates what the code will do."
"You then produce exactly as much code as necessary to pass the test. Then you refactor (simplify and clarify) both production code and test code"

Test-driven development is a programming technique that requires you to write actual code and automated test code simultaneously. This ensures that you test your code — and enables you to retest your code quickly and easily, since it's automated.

Origin of TDD

- Emerged from Agile and eXtreme
- Programming (XP) methods
- XP Practices
  - Incremental
  - Continuous Integration
  - Design Through Refactoring
  - Collective Ownership
  - Programmer Courage
- Lightweight development process
- K. Beck: "XP takes best practices and turns all knobs up to 10!"

Testing Paradigms

Waterfall — test last

Analysis $\Rightarrow$ Implementation $\Rightarrow$ Testing

Concurrent Testing — independent testing

$$\text{Analysis} \Rightarrow \begin{cases} \Rightarrow \text{Implementation} \\ \qquad \Rightarrow \text{Testing} \end{cases}$$

Agile — test first

Analysis $\Rightarrow$ Testing $\Rightarrow$ Implementation

What is TDD

- TDD is a technique whereby you write your test cases before you write any implementation code
- Tests drive or dictate the code that is developed
- An indication of *intent*
  - Tests provide a *specification* of what a piece of code actually does
  - Thinking about testing is analysing what the system should do!
  - Some might argue that "tests are part of the documentation"
- Mainly Unit Testing
- Automated Regression Unit Testing

Automated Testing

- Code that isn't tested doesn't work
- Code that isn't regression tested suffers from code rot (breaks eventually)
- If it is not automated it is not done!
- A unit testing framework enables efficient and effective unit and regression testing
- Programmer Friendly

Regression Testing

- New code and changes to old code can affect the rest of the code base. "Affect" sometimes means "break"
- Regression means "Relapsed to a less perfect or developed state"
- Regression testing: Check that code has not regressed
- Regression testing is required for a stable, maintainable code base

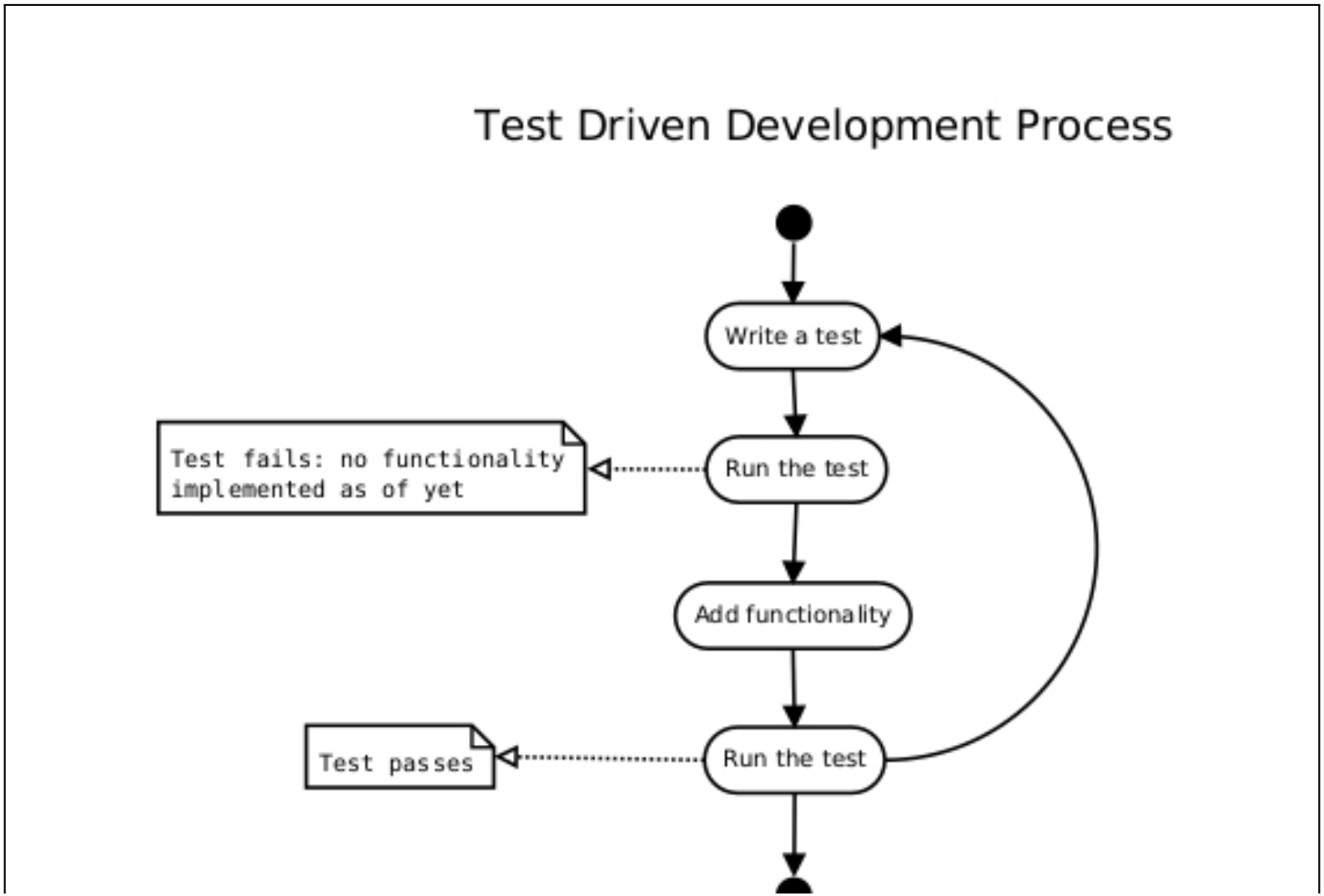
Refactoring

- Refactoring is a behavior preserving transformation
- Restructure, simplify, beautify
- Refactoring is an excellent way to break code.

TDD Phases

1. Write a single test
2. Compile it. It shouldn't compile because you've not written the implementation code
3. Implement just enough code to get the test to compile
4. Run the test and see it fail
5. Implement just enough code to get the test to pass
6. Run the test and see it pass
7. Refactor for clarity and once and only once
8. Repeat

TDD Process


Test Driven Development Process
Write a test
Run the test
Test fails: no functionality implemented as of yet
Add functionality
Run the test
Test passes

TDD Example

```
float dep = 1.01;
check (ba.balance == dep); //test fails
//check bank deposit function
BankAccount ba = new BankAccount;
check (ba != NULL);
ba.deposit(dep);
check (ba.balance == dep); //test passes
```

TDD Question

How is TDD different from your usual development?

Benefits of TDD

- Efficiency
  - Identify defects earlier
  - Identify cause more easily
- Higher value of test effort
  - Producing a more reliable system
  - Improve quality of testing (maintain automated tests)
  - Minimization of schedule
- Reducing Defect Injection
  - Small fixes have are 40 times more error prone than new code
  - $\Rightarrow$ Fine grained tests + run tests continuously

Benefits

- Better programmer Life
  - Can now work on your code with no fear;
  - No one wants to support a fragile system;
  - "We don't touch that, it might break." With complete tests, code away:
  - Test fails, you *know* you broke something.
  - Tests pass, you didn't.
- Eases changes (XP embrace change):
  - addition of functionality
  - new requirements
  - refactoring

Empirical Studies Show Test Driven Development Improves Quality

`www.infoq.com/news/2009/03/TDD-Improves-Quality`
A paper first published in the Empirical Software Engineering journal reports: "TDD seems to be applicable in various domains and can significantly reduce the defect density of developed software without significant productivity reduction of the development team." The study compared 4 projects, at Microsoft and IBM that used TDD with similar projects that did not use TDD.

Quantitatively Evaluating Test-Driven Development

`http://www.nomachetejuggling.com/files/tdd_thesis.pdf`

This study provided substantial evidence that Test-Driven Development is, indeed, an effective tool for improving the quality of source code. Test-Driven Development offered an overall improvement of 21% over code written using the test-last technique. Considering the sheer volume of the codebases being analyzed, its reasonable to conclude that the substantial difference is noteworthy.

Test-Driven Development was most effective at decreasing the complexity of the code, with an improvement of 31%.

The next-best improvement was on cohesion metrics, an improvement of 21%.

The smallest improvement Test-Driven Development offered was on coupling metrics, only 10%.

A comparison of the efficacy of test-driven learning versus self-assessment learning

www.ncbi.nlm.nih.gov/pmc/articles/PMC3791901/
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.

Questions

1. What are the two basic activities of test driven development?
3. What are six steps we can identify as a practice of test driven development
9. What is refactoring?
10. Why do we refactor code?
11. What happens if a test passes?
12. What happens if a test fails?

Answers

1. Use automated tests
2. Write the tests before you write the implementation code
3. Write an automated test
4. Run the test — it fails
5. Write the implementation
6. Run the test — it passes
7. Refactor once (and only once)
8. Run the test — it passes
9. Refactoring is behavior preserving code transformation
10. To simplify and restructure code to make it more efficient and easier to maintain
11. You are done if a test passes
12. If a test fails, you write more code