Test Driven Development

Brian Nielsen
Arne Skou

bnielsen@cs.aau.dk
ask@cs.aau.dk
TDD Definition

“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”

[Mansel&Husted: JUnit in Action]
TDD Definition [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.”
Possible test processes

Test Last (waterfall)
- Specification
- Code Impl.
- Write Test cases

Test Concurrently (independently)
- Specification
- Write Test cases
- Code Impl.

Test First
- Specification
- Write Test cases
- Code Impl.
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
Requirements

Standards

Informal understanding in developer’s mind

Written specs (informal, incomplete, ambiguous)

Domain Experts

Customers
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 & 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** = 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.
Testing using xUnit

All tests pass

Refactor code tests [Pass]

Refactor code tests [Fail]

Fix code

Can’t think of any more tests

Write Failed test

Some tests fail
Benefits?

- **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
  - Stable code base

- **Reducing Defect Injection**
  - Small “fixes” have are 40 times more error prone than new code => 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
TDD Stages

In Extreme Programming Explored (The Green Book), Bill Wake describes the test / code cycle:

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
Development Cycle

1. Add a test
2. Run the test
   - [Fail] Add functionality
     - [Fail] Run the test
       - [Pass] Development continues
       - [Pass] Development stops
TDD Example
Simple Light-Controller

- Light controller toggles light on/off when wire is touched
TDD Example

- Writing test case first

```c
void testSwitch() {
    s = new LightSwitch();
    check(s != NULL);
    check(ON == s.switch());
    check(OFF == s.switch());
    check(ON == s.switch());
}
```

- Run tests: (Fails: compilation errors)
- LightSwitch doesn’t exist
TDD Example

• Write a first simple implementation

```java
public enum LightState {ON, OFF} state;
public LightSwitch() {state=OFF; }

LightState switch() {
    return state;
}
```

• Run Tests
  • System Compiles
  • Test still fails (passes first check)
  • Switch not fully implemented
TDD Example

• Implement switch-method

```java
public Class Class LightSwitch {
    public enum LightState {ON, OFF} state;
    public LightSwitch() {state=OFF; }

    LightState switch() {
        if(state==OFF) state=ON;
        if(state==ON) state=OFF;
        return state;
    }
}
```

• Run Test
  • still fails (passes first two checks)
  • Switch incorrect
TDD Example

- Rewrite `switch-method` (perhaps refactor)

```java
Class LightSwitch {
    public enum LightState {ON, OFF} state;
    public LightSwitch() {state=OFF; }

    public LightState switch() {
        if(state==ON)
            state=OFF;
        else
            state=ON;

        return state;
    }
}
```

- Run Tests: Test Passes
TDD Example

- Light controller toggles light on/off when wire is touched
- **New Requirement:** When wire is held the controller decrements the light level
TDD Example

• Add test case for new functionality
TDD Example

• Write Implementation

```java
class LightSwitch {
    public enum LightState {ON, OFF} state;
    public LightSwitch(){state=OFF; }
    int level;
    public int getLevel(){return level ; }
    public void dim(){
        if(state==ON && level>1) level --;
    }
    public LightState switch(){
        if(state==ON) {
            // changed code
            state=ON;
            level =0;
        } else {
            state=ON;
            level =3;
        }
        return state;
    }
}
```

• Run Tests: `testDim` Passes, but `testSwitch` fail
TDD Example

- Fix Error

```java
public enum LightState {ON, OFF} state;
public LightSwitch(){state=OFF;}
int level;
public int getLevel(){return level;}
public void dim(){
    if(state==ON && level>1) level--;
}
public LightState switch(){
    if(state==ON) {
        //changed code
        state=OFF;
        level =0;
    } else {
        state=ON;
        level =3;
    }
    return state;
}
```

- Run Tests: Both `testSwitch` and `testDim` passes
A Case Study

Device Drivers at IBM [Williams, Maximilien, Vouk ’03]

<table>
<thead>
<tr>
<th></th>
<th>Legacy 7th Iteration</th>
<th>New 1st Release</th>
</tr>
</thead>
<tbody>
<tr>
<td>Team Size (Developers)</td>
<td>5</td>
<td>9</td>
</tr>
<tr>
<td>Team Experience</td>
<td>Experienced</td>
<td>Some Inexperienced</td>
</tr>
<tr>
<td>(Language and Domain)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Collocation</td>
<td>Collocated</td>
<td>Distributed</td>
</tr>
<tr>
<td>Code Size (KLOC)</td>
<td>6.6; 49.9; 56.5</td>
<td>64.6; 9.0; 73.6</td>
</tr>
<tr>
<td>New; Base; Total</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Language</td>
<td>Java/C++</td>
<td>Java</td>
</tr>
<tr>
<td>Unit Testing</td>
<td>Ad hoc</td>
<td>TDD</td>
</tr>
<tr>
<td>Technical Leadership</td>
<td>Shared resource</td>
<td>Dedicated coach</td>
</tr>
</tbody>
</table>

- None experienced in TDD
Results

- 40% reduction in defect density (external test team)
- Identical severity distribution

- Approximately same productivity
  - Developers spend more time writing test cases, but reduces time spent on (unpredictable) debugging
  - 64.6 KLOC new code + 34 KLOC JUnit tests
- “We believe that TDD aided us in producing a product that more easily incorporated later changes”
Background for 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!”
Books

test-driven development: A Practical Guide
Dave Astels
Prentice-Hall/Pearson Education, 2003
ISBN 0-13-101649-0
Reviewed BUG developers’ magazine, Nov/Dec 2003

Test-Driven Development: By Example
Kent Beck
Addison-Wesley, 2003
ISBN 0-321-14653-0
Resources (Books)

Refactoring: Improving the Design of Existing Code
Martin Fowler
Addison-Wesley, 1999
References and links

- S. Amber. Introduction to Test Driven Development (TDD). [www.agiledata.org](http://www.agiledata.org)
- [http://www.testdriven.com](http://www.testdriven.com)
- [http://www.junit.org](http://www.junit.org)
Summary

TDD
  =
 Test first
  +
 Automated (Unit) Testing

RED
GREEN
REFACTOR
GREEN