Test Driven Development

Brian Nielsen
Arne Skou

bnielsen@cs.aau.dk
ask@cs.aau.dk

BRICS
Basic Research
in Computer Science

CISS
CENTER FOR INDLEJREDE SOFTWARE SYSTEMER

TDD Definifion

“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 Definifion [Agile Alliance]

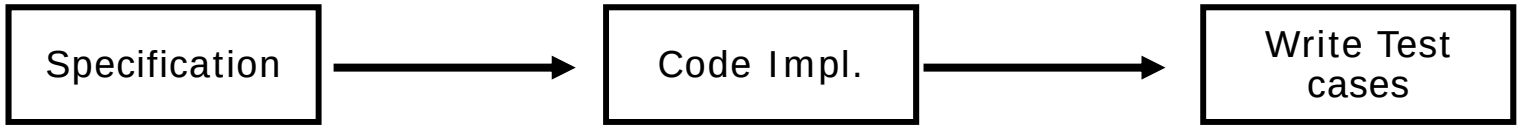
“Test Driven Development is the craft of producing automated tests for production code, and using that process to drive design and programming

For every bit of functionality, you first develop a test that specifies and validates what the code will do.

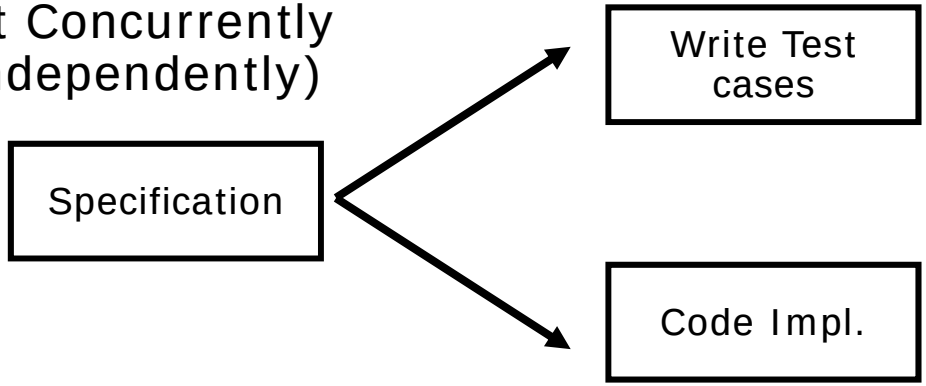
You then produce exactly as much code as necessary to pass the test. Then you refactor (simplify and clarify) both production code and test code”

Possible test processes

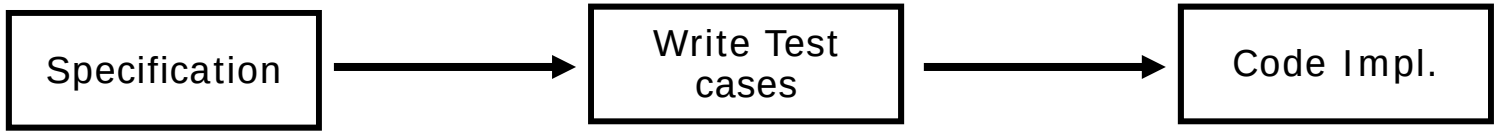
Test Last (waterfall)



Test Concurrently (independently)



Test First

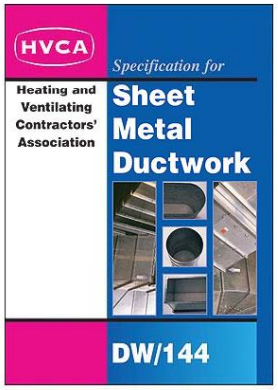


CISS

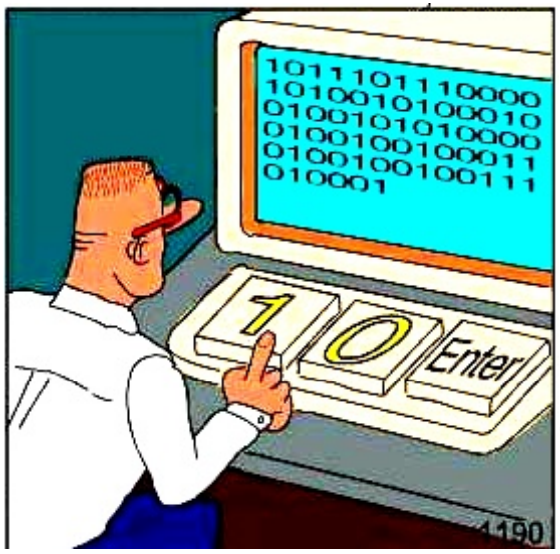
What is TDD?

- 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

Domain Experts

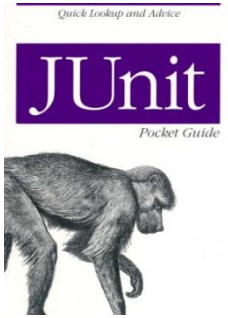
Written specs
(iinformal, ncomplete, ambiguous)

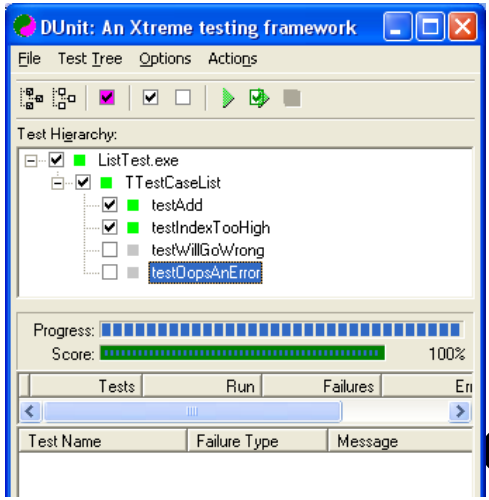
Customers

CISS

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





CISS

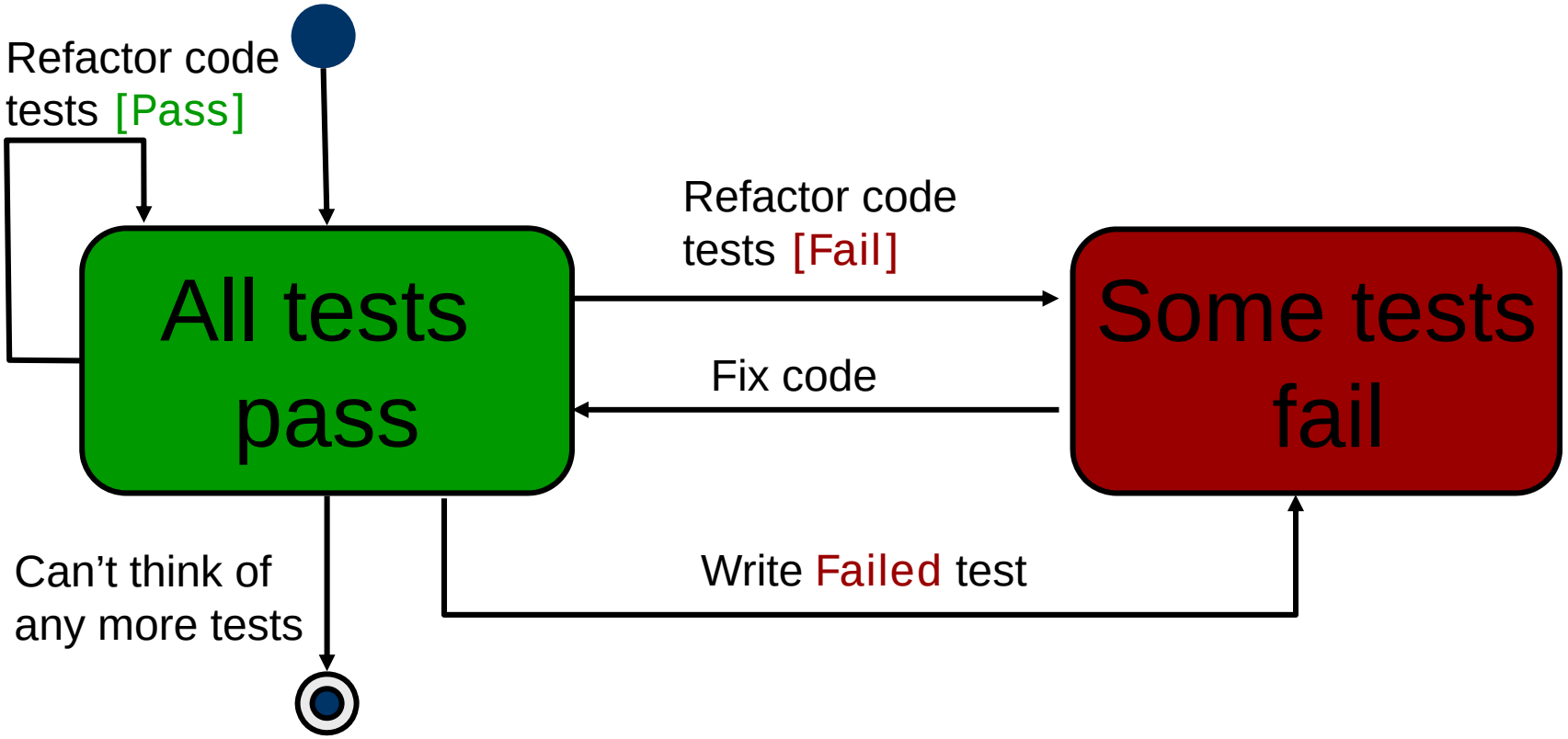
Regression testing

- 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



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

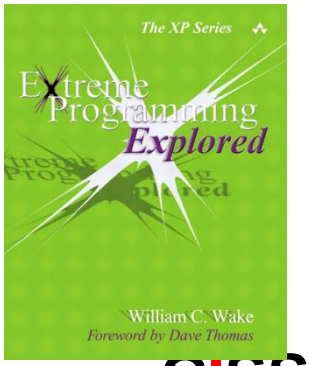
Benefits?

- 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

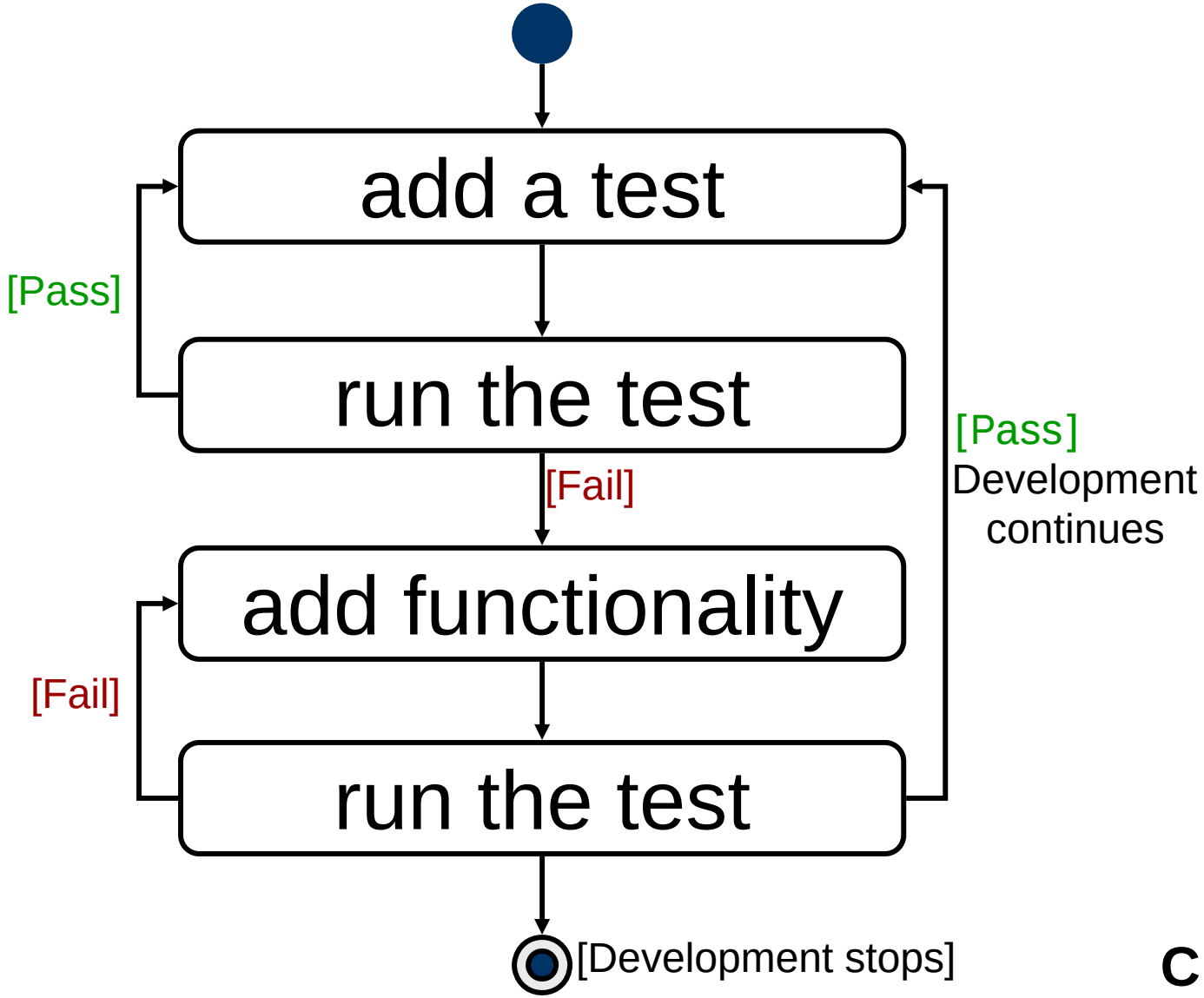
- In Extreme Programming Explored (The Green Book), Bill Wake describes the test / code cycle:

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



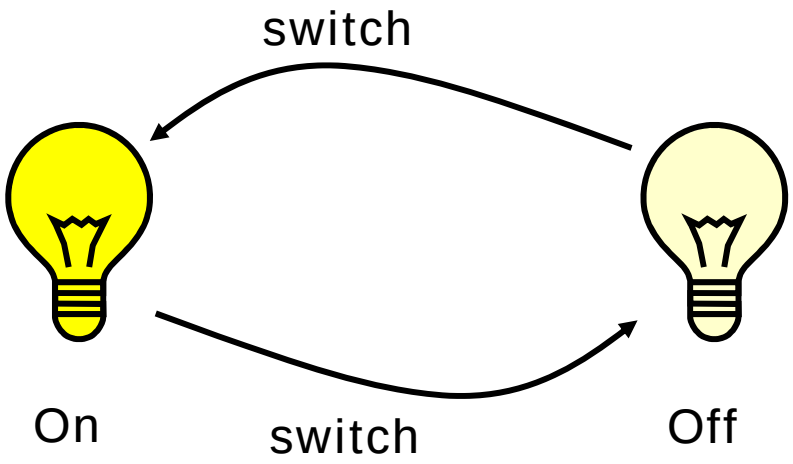
CISS

Development Cycle



TDD Example
Simple Light-Controller

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



TDD Example

- Writing test case first

```
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

```
Class LightSwitch {
  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

```
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

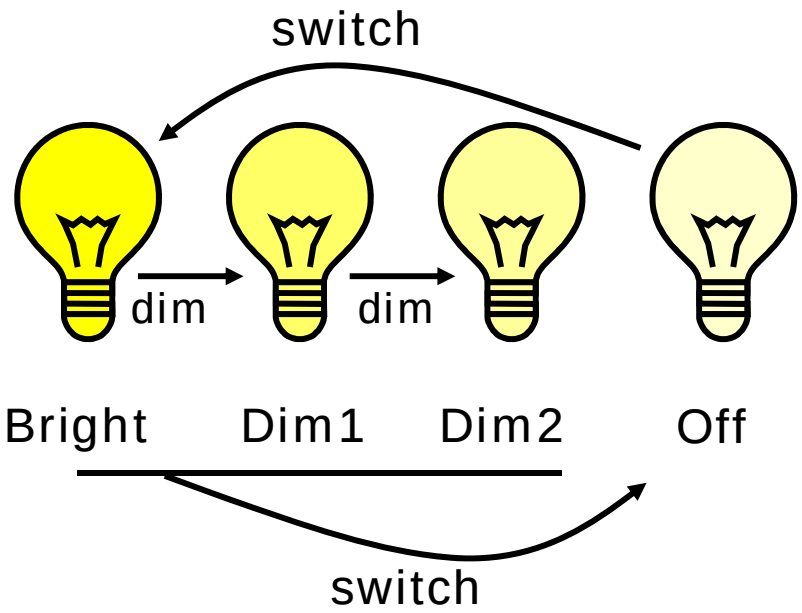
- Rewrite switch-method (perhaps refactor)

```
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

```
void testSwitch() {
 s=new LightSwitch();
check(s!=NULL);
check(ON == s.switch());
check(OFF ==s.switch());
check(ON == s.switch());
}
void testDimmer(){
 s=new LightSwitch(); //initially off
 check(s.getLevel()==0);
 s.dim();              //No effect when off
 check(s.getLevel()==0);
 s.switch();           //switch on: level is Max:3
 check(s.getLevel()==3);
 s.dim();
 check(s.getLevel()==2); //dimming works
 s.dim();
 check(s.getLevel()==1);
 s.dim();
 check(s.getLevel()==1); //cannot dim more
}
```

TDD Example

- Write Implementation

```
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

```
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=OFF;
       level=0;
     } else {
       state=ON;
       level=3;
     }
     return state;
}}
```

- Run Tests: Both ***testSwitch*** and ***testDim*** passes

A Case Study

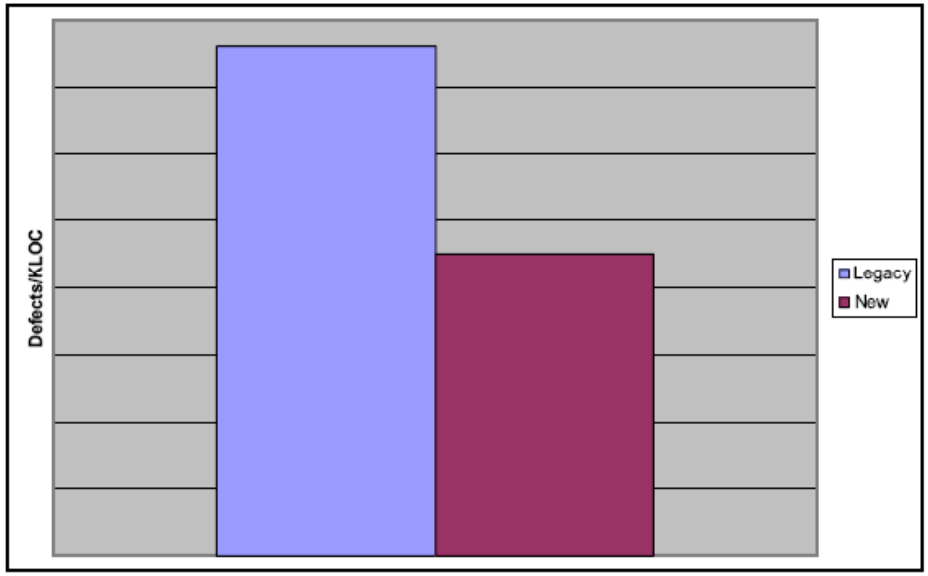
- Device Drivers at IBM [Williams, Maximilien, Vouk '03]

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

- None experienced in TDD

Results

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



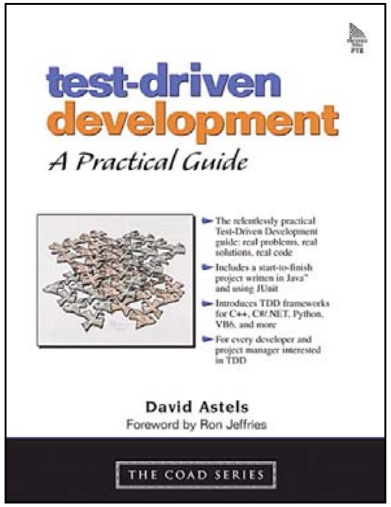
- Approximately same productivity
  - Developers spend more time writing test cases, but reduces time spent on (unpredictable) debugging
  - 64.6 KLOC new code + 34 KLOC JUnit tests
- *"We belive that TDD aided us in producing a product that more easily incorporated later changes"*

CISS

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



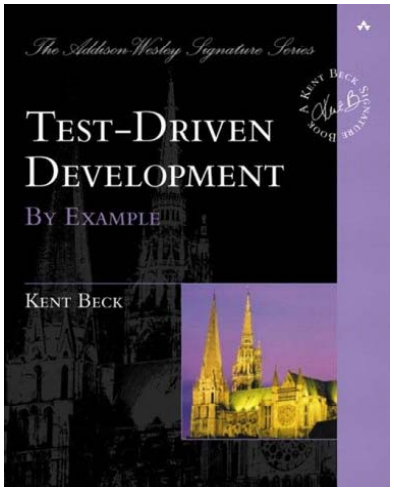
test-driven development: A Practical Guide

Dave Astels

Prentice-Hall/Pearson Education, 2003

ISBN 0-13-101649-0

Reviewed BUG developers' magazine, Nov/Dec 2003



Test-Driven Development: By Example

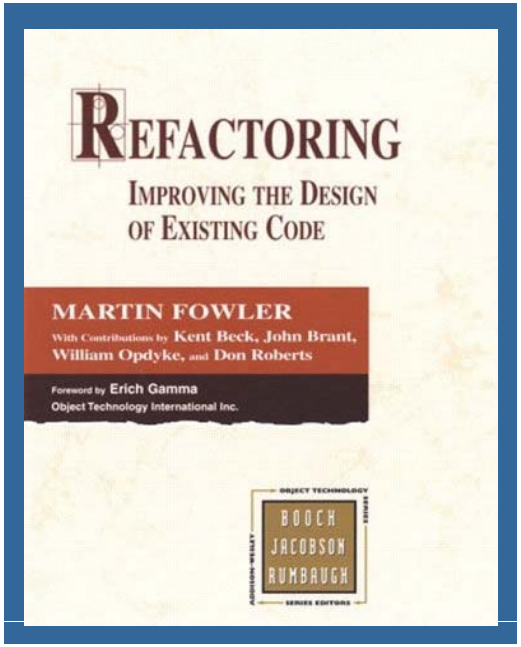
Kent Beck

Addison-Wesley, 2003

ISBN 0-321-14653-0

CISS

Resources (Books)



Refactoring: Improving the Design of Existing Code

Martin Fowler
Addison-Wesley, 1999
ISBN 0-201-48567-2

CISS

References and links

- S. Amber. Introduction to Test Driven Development (TDD). www.agiledata.org
- D. Jansen and H. Saiedian. Test-Driven Development: Concepts, Taxonomy and Future Direction. *IEEE Computer September 2005*
- E. M. Maximilien and L. Williams. Assessing Test-Driven Development at IBM. *25th International Conference on Software Engineering*, 2003
- K. Beck and E. Gama. Test infected: Programmers love writing tests. *Java Report*, 3(7), July 1998
- http://www.testdriven.com
- http://www.junit.org

CISS

Summary

TDD

=

Test first

+

Automated (Unit) Testing

RED

GREEN

REFACTOR

GREEN

CISS

End

BRICS
Basic Research
in Computer Science