



NRC-CNRC

*Institute for
Information
Technology*

An Overview of Test-Driven Development

Hakan Erdogmus

October 18, 2007



National Research
Council Canada

Conseil national
de recherches Canada

Canada

Outline

- **What is TDD? How does it work?**
- **TDD dynamics**
- **Empirical results**
- **TDD controversy & myths**
- **Supporting tools and frameworks**
- **Challenges & Conclusions**

What is TDD?

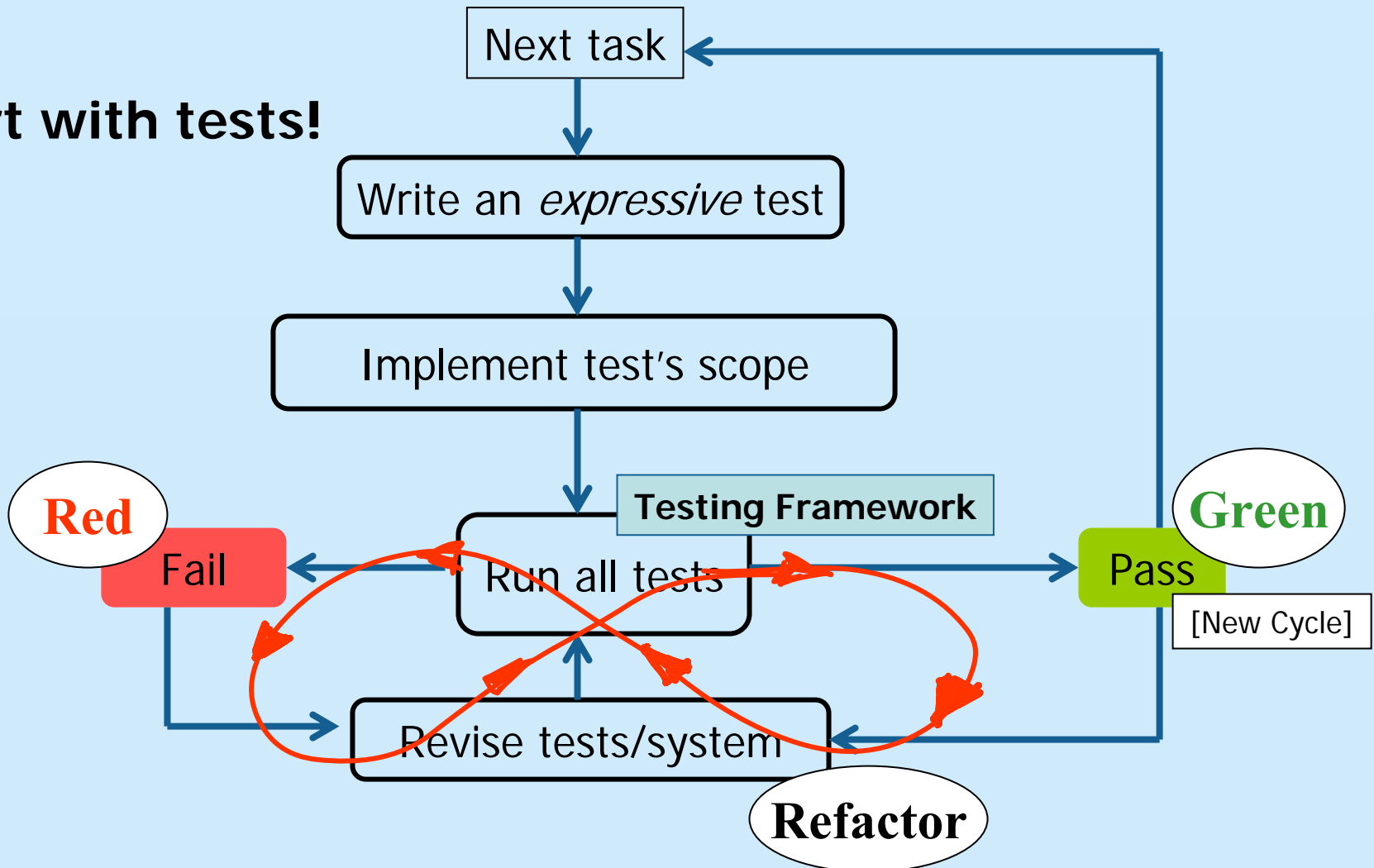
An **incremental** software development approach
...that relies on **automated regression tests**
...to **steer** development activities

- Popularized by Extreme Programming (Beck, Cunningham, Martin, Astels, ...)
- Can be used at different levels:
 - System/Team: Customers' **acceptance tests** drive **development of new features**
 - Module/Personal: Programmers' **unit tests** drive **task implementation**

How does it work?

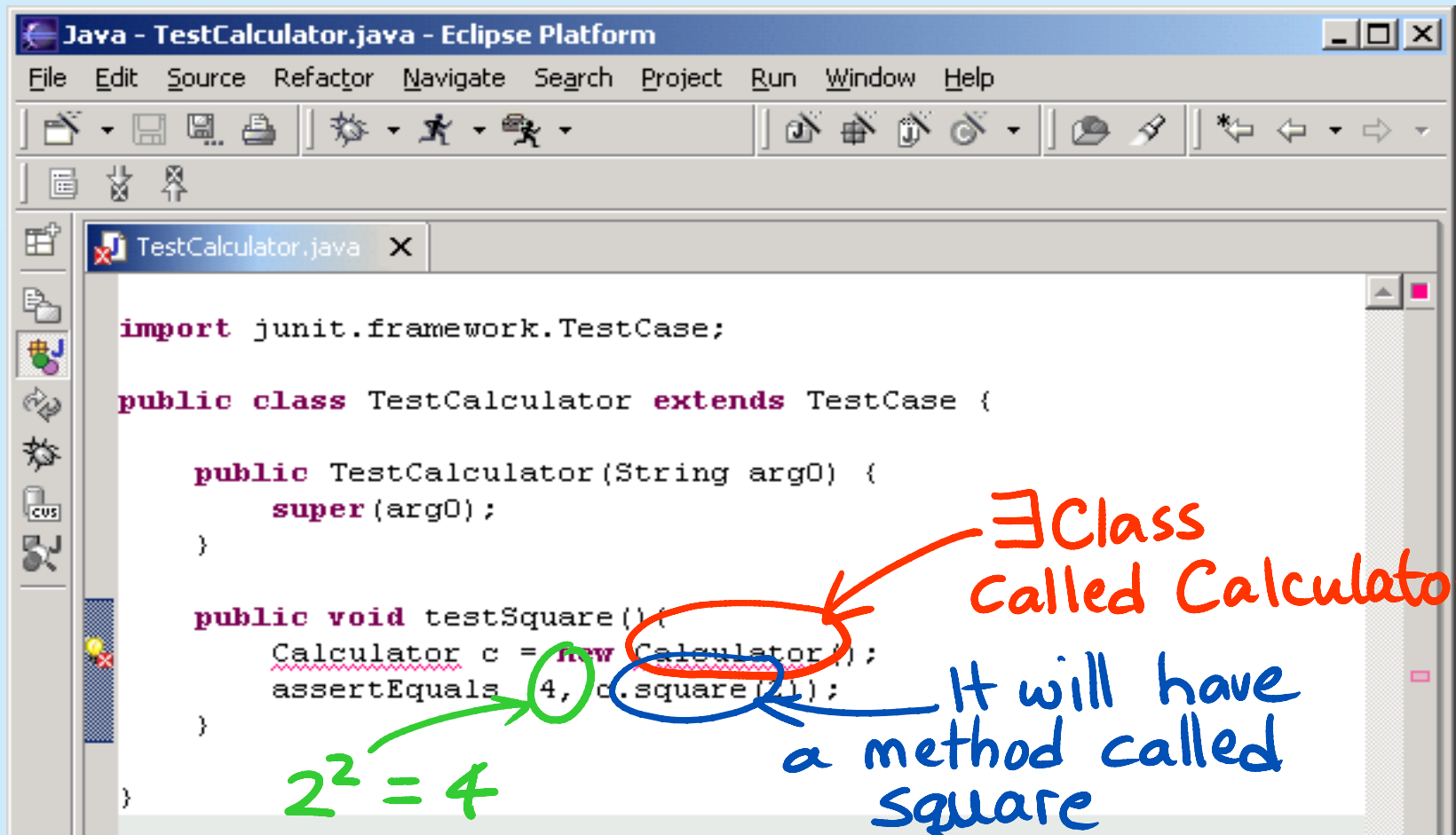
Tiny little steps...

Start with tests!



Example - Task implementation with JUnit

[Begin Cycle 1] Write a test



The screenshot shows the Eclipse IDE with a Java file named `TestCalculator.java`. The code is as follows:

```
import junit.framework.TestCase;

public class TestCalculator extends TestCase {

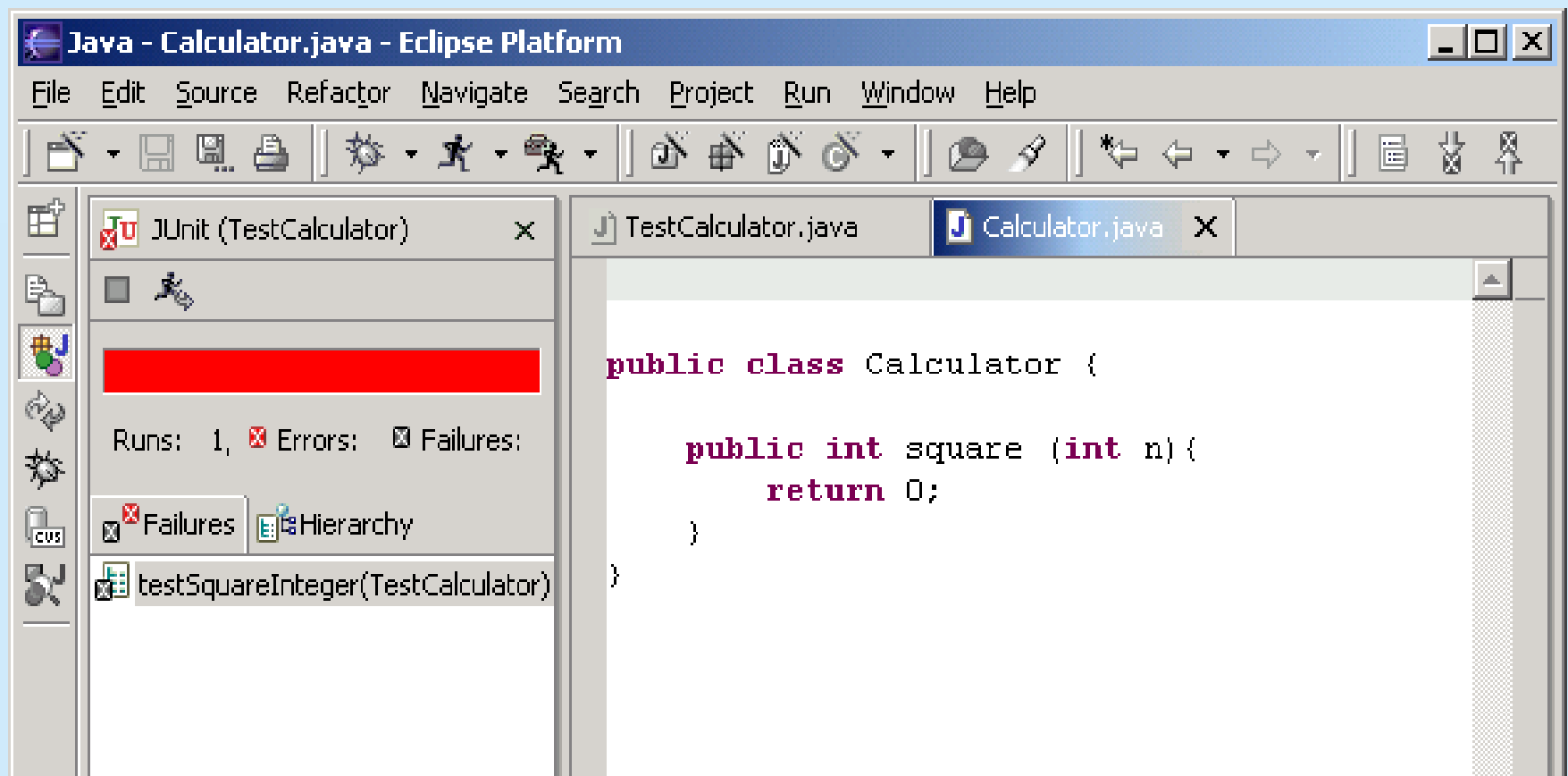
    public TestCalculator(String arg0) {
        super(arg0);
    }

    public void testSquare() {
        Calculator c = new Calculator();
        assertEquals(4, c.square(2));
    }
}
```

Handwritten annotations in red, blue, and green are present:

- A red arrow points from the text "Class called Calculator" to the `Calculator` class name in the `new Calculator()` instantiation.
- A blue arrow points from the text "It will have a method called square" to the `square(2)` method call.
- A green arrow points from the text " $2^2 = 4$ " to the `assertEquals(4, ...)` assertion.

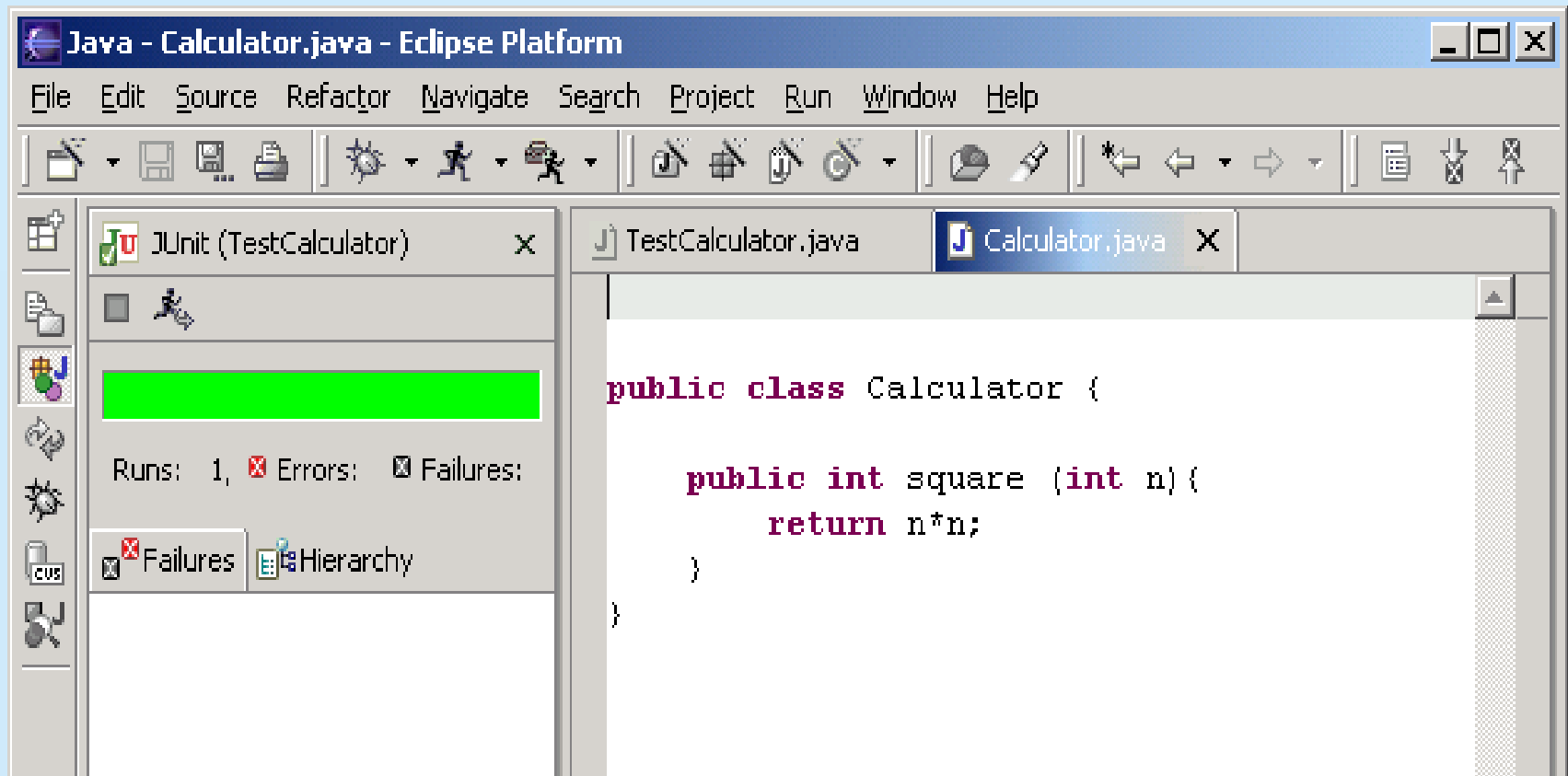
Let the test fail



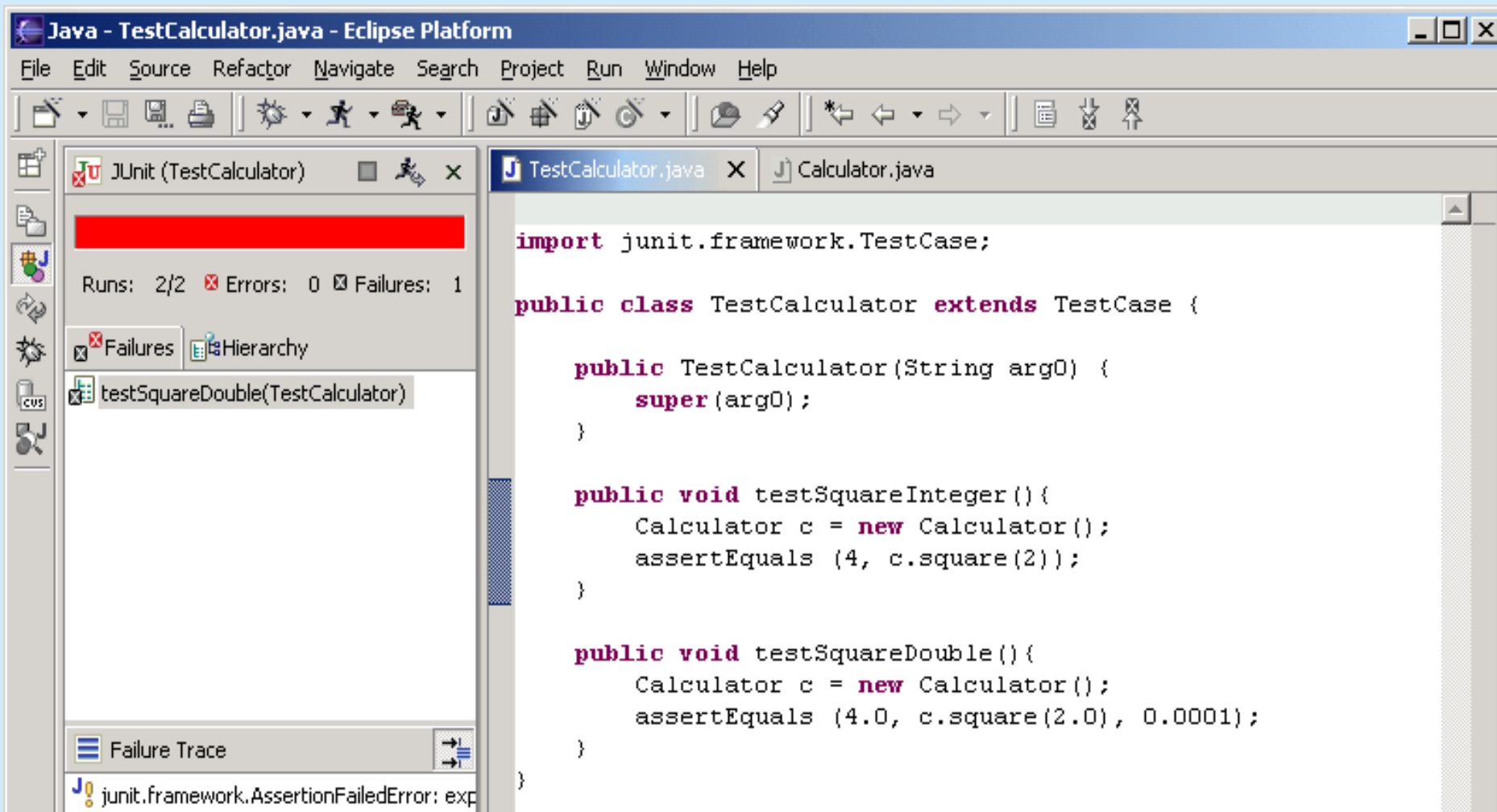
The screenshot shows the Eclipse IDE interface. The title bar reads "Java - Calculator.java - Eclipse Platform". The menu bar includes File, Edit, Source, Refactor, Navigate, Search, Project, Run, Window, and Help. The toolbar contains various icons for file operations, debugging, and navigation. The left sidebar shows the "JUnit (TestCalculator)" view with a red bar indicating a failure. Below the red bar, it displays "Runs: 1, Errors: 0, Failures: 1". The "Failures" tab is selected, showing a list of failures: "testSquareInteger(TestCalculator)". The main editor window shows the code for the "Calculator" class:

```
public class Calculator {  
  
    public int square (int n){  
        return 0;  
    }  
}
```

Make the test pass [End Cycle 1]



[Begin Cycle 2] Improve the function – but write a test first



The screenshot shows the Eclipse IDE interface. The main editor displays the source code for `TestCalculator.java`, which extends `TestCase` and contains three test methods: `testSquareInteger()` and `testSquareDouble()`. The `testSquareInteger()` method asserts that `Calculator.square(2)` equals 4. The `testSquareDouble()` method asserts that `Calculator.square(2.0)` equals 4.0 with a precision of 0.0001. The `testSquareDouble()` method is currently selected in the editor.

The left sidebar shows the JUnit test runner. The top bar indicates "Runs: 2/2", "Errors: 0", and "Failures: 1". Below this, the "Failures" view shows a tree structure with `testSquareDouble(TestCalculator)` expanded and marked as failed. The "Failure Trace" at the bottom shows the error message: `junit.framework.AssertionFailedError: exp`.

```
import junit.framework.TestCase;

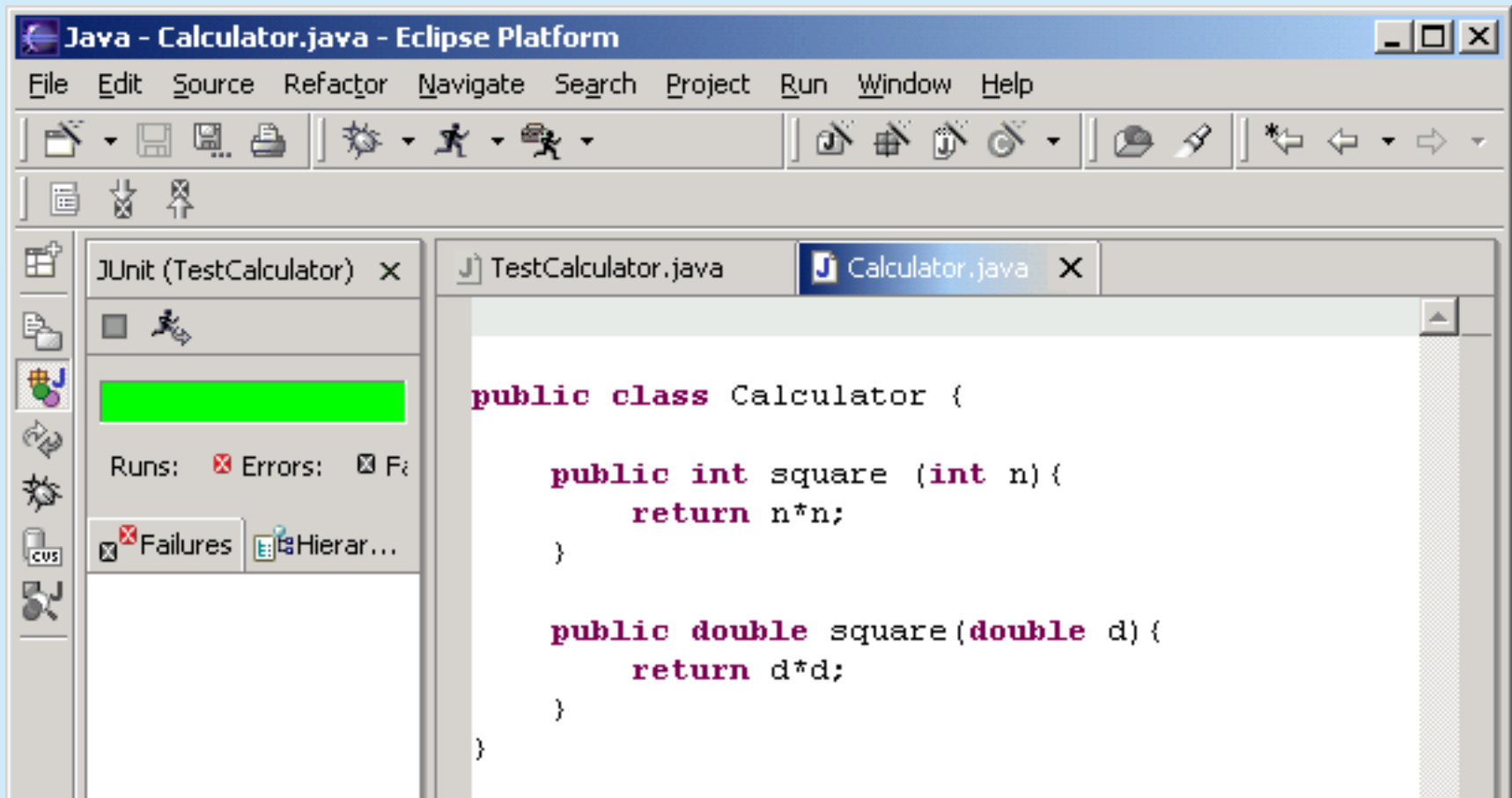
public class TestCalculator extends TestCase {

    public TestCalculator(String arg0) {
        super(arg0);
    }

    public void testSquareInteger(){
        Calculator c = new Calculator();
        assertEquals (4, c.square(2));
    }

    public void testSquareDouble(){
        Calculator c = new Calculator();
        assertEquals (4.0, c.square(2.0), 0.0001);
    }
}
```

Make both tests pass [End Cycle 2]



The screenshot shows the Eclipse IDE interface. The title bar reads "Java - Calculator.java - Eclipse Platform". The menu bar includes File, Edit, Source, Refactor, Navigate, Search, Project, Run, Window, and Help. The toolbar contains various icons for file operations and development tools. The left sidebar shows the "JUnit (TestCalculator)" view with a green bar and a "Failures" tab selected. The main editor displays the following code:

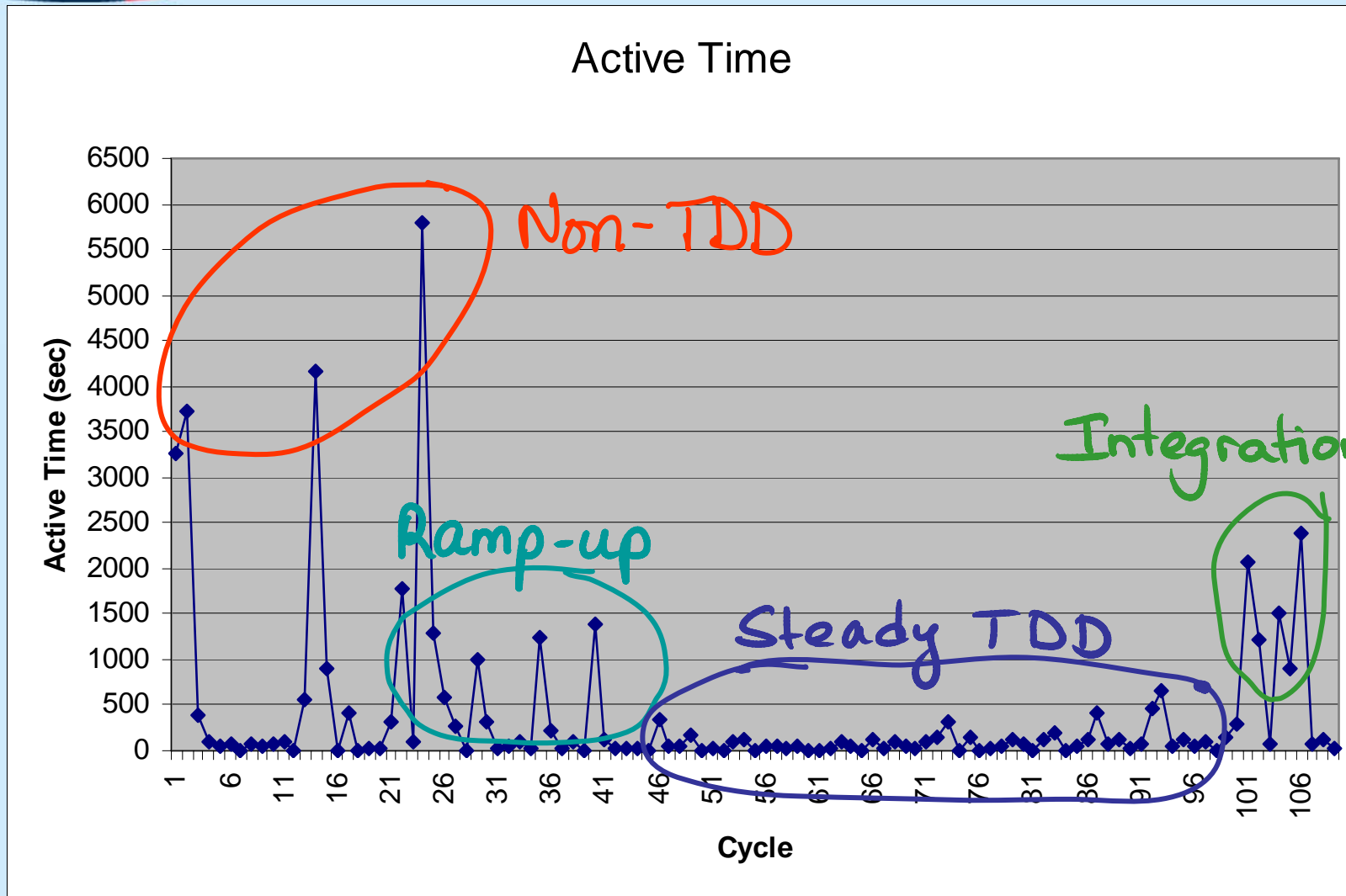
```
public class Calculator {  
  
    public int square (int n){  
        return n*n;  
    }  
  
    public double square (double d){  
        return d*d;  
    }  
}
```

TDD Dynamics

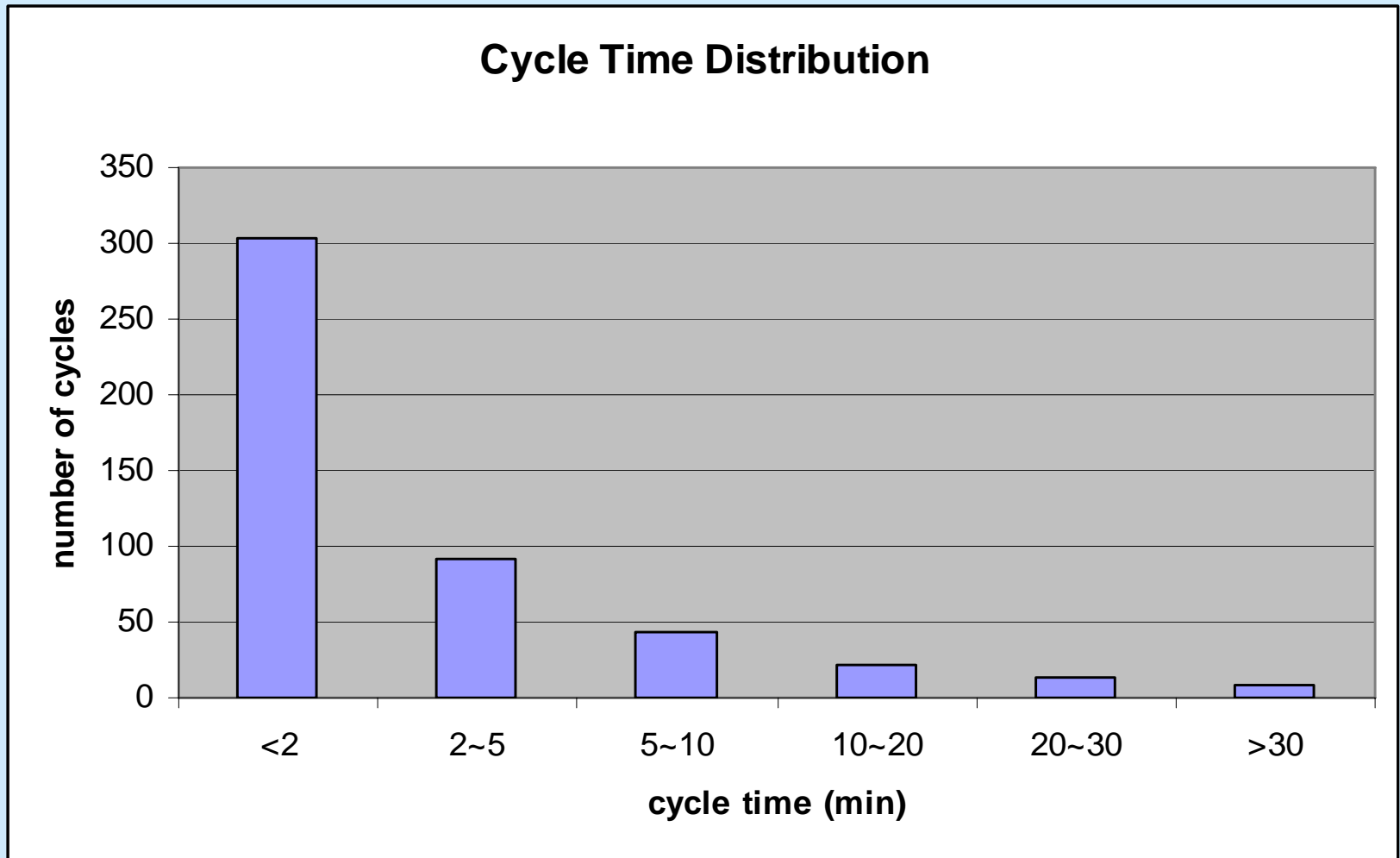
Task implementation with JUnit tests

- How long are the cycles?
- How much test code is written?
- How much effort is spent on testing?
- Progression to full TDD

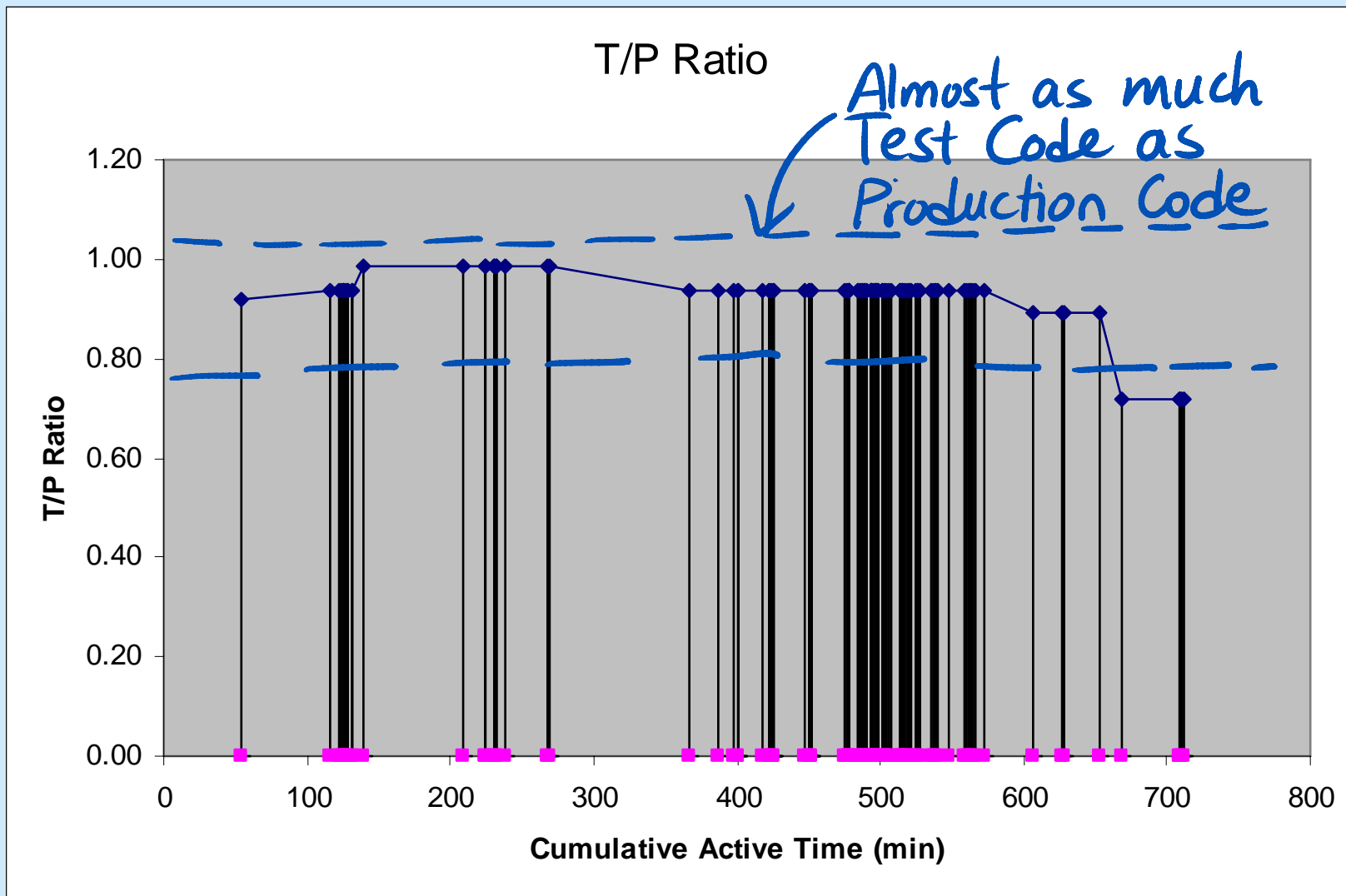
TDD Dynamics: Cycle Time



TDD Dynamics: Cycle Time Distribution

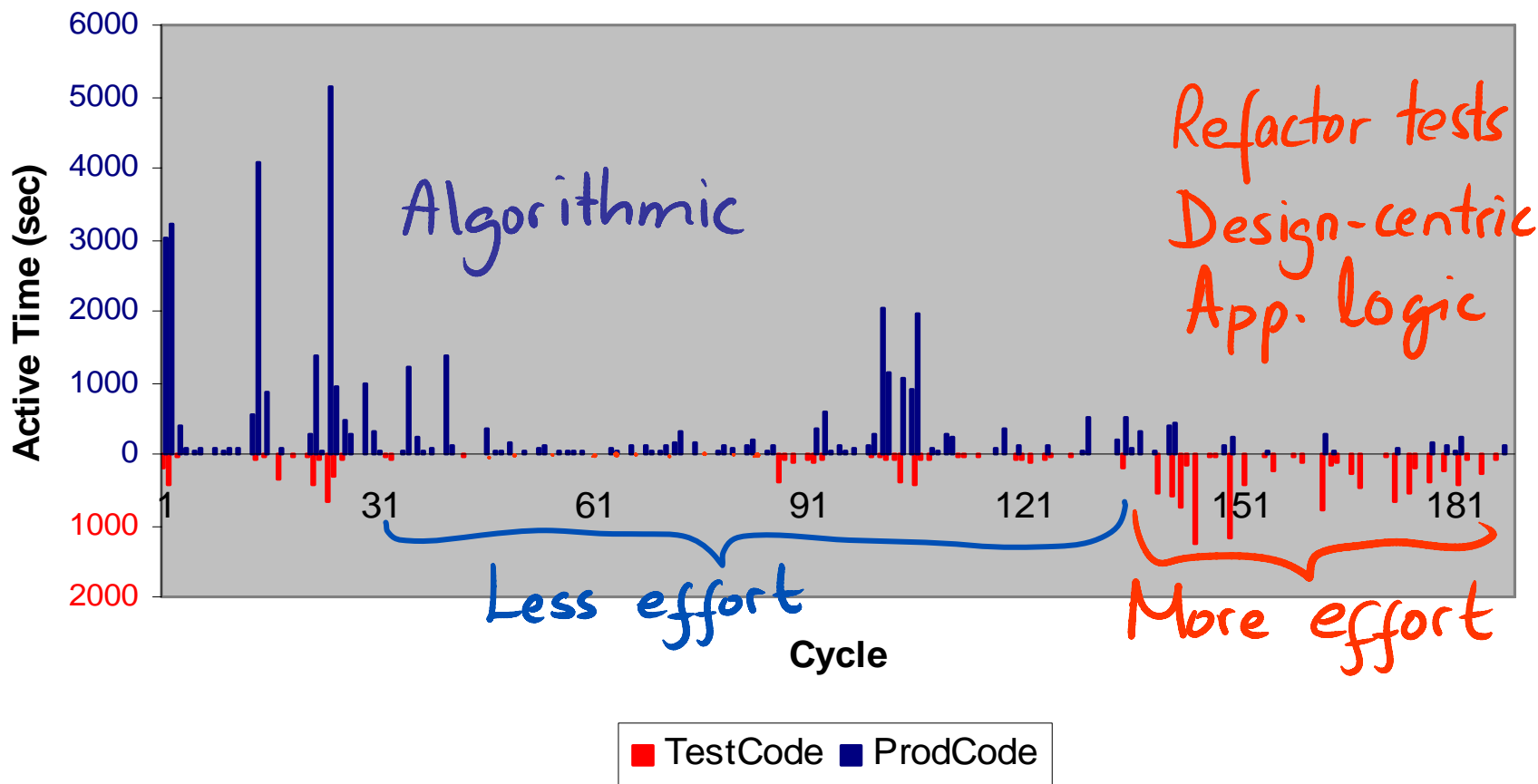


TDD Dynamics: Test Code Size



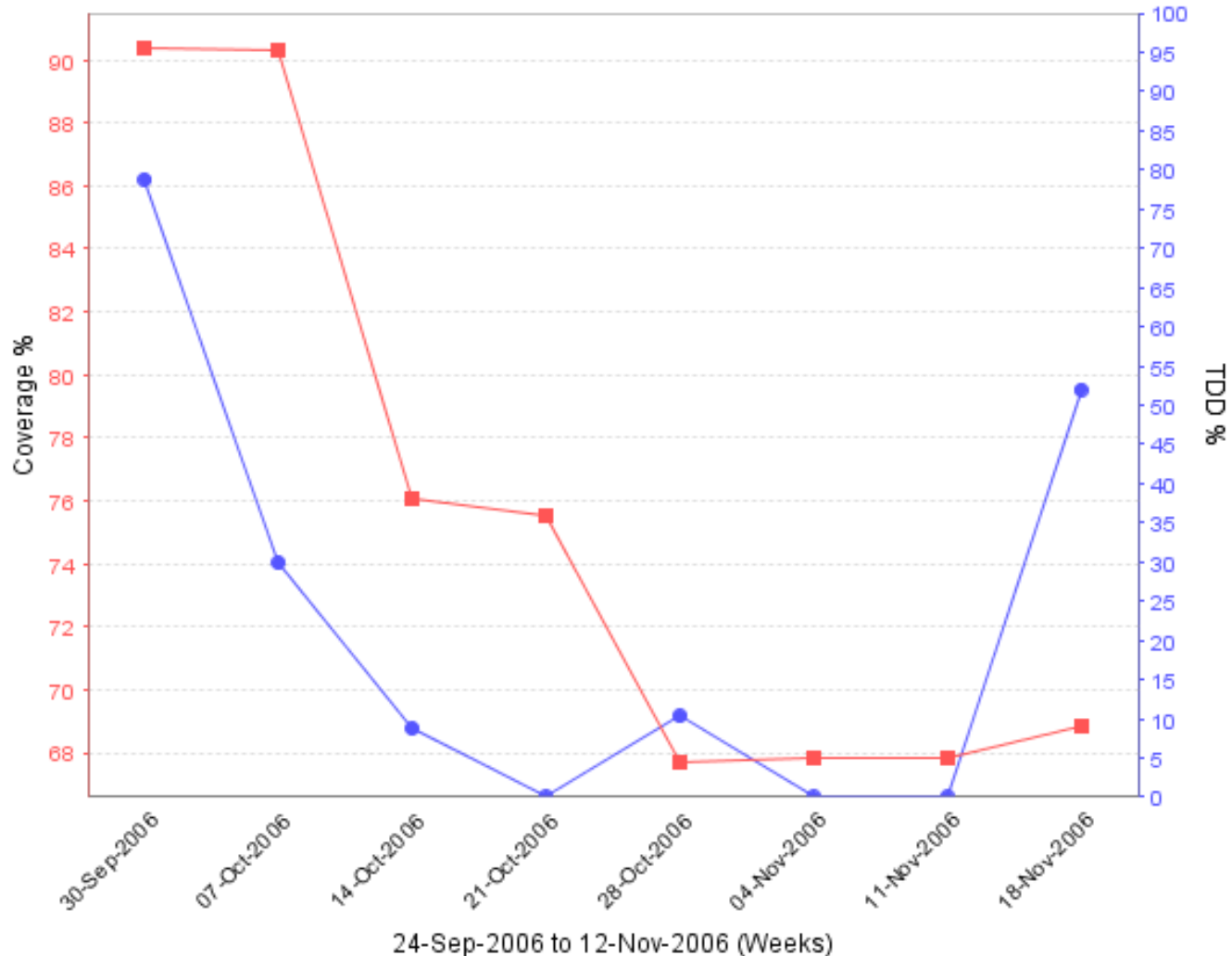
TDD Dynamics: Testing Effort

Testing Effort



TDD Dynamics: Test Coverage

Percentage of TDD Episodes (time) and Coverage

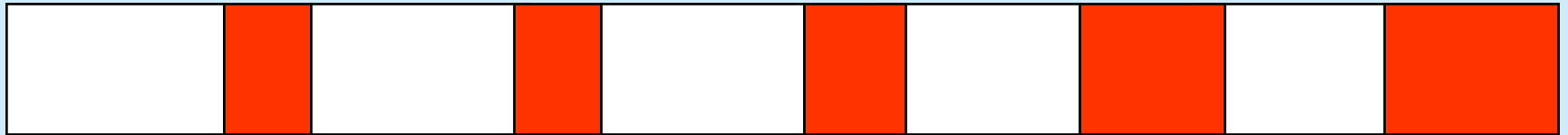
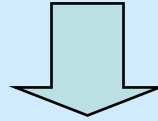


■ Coverage-Percentage<**, line> ● TDDPercent<time>:

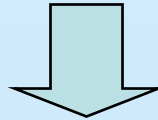
TDD Dynamics: Progression



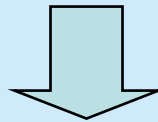
Traditional



Iterative – Non-TD

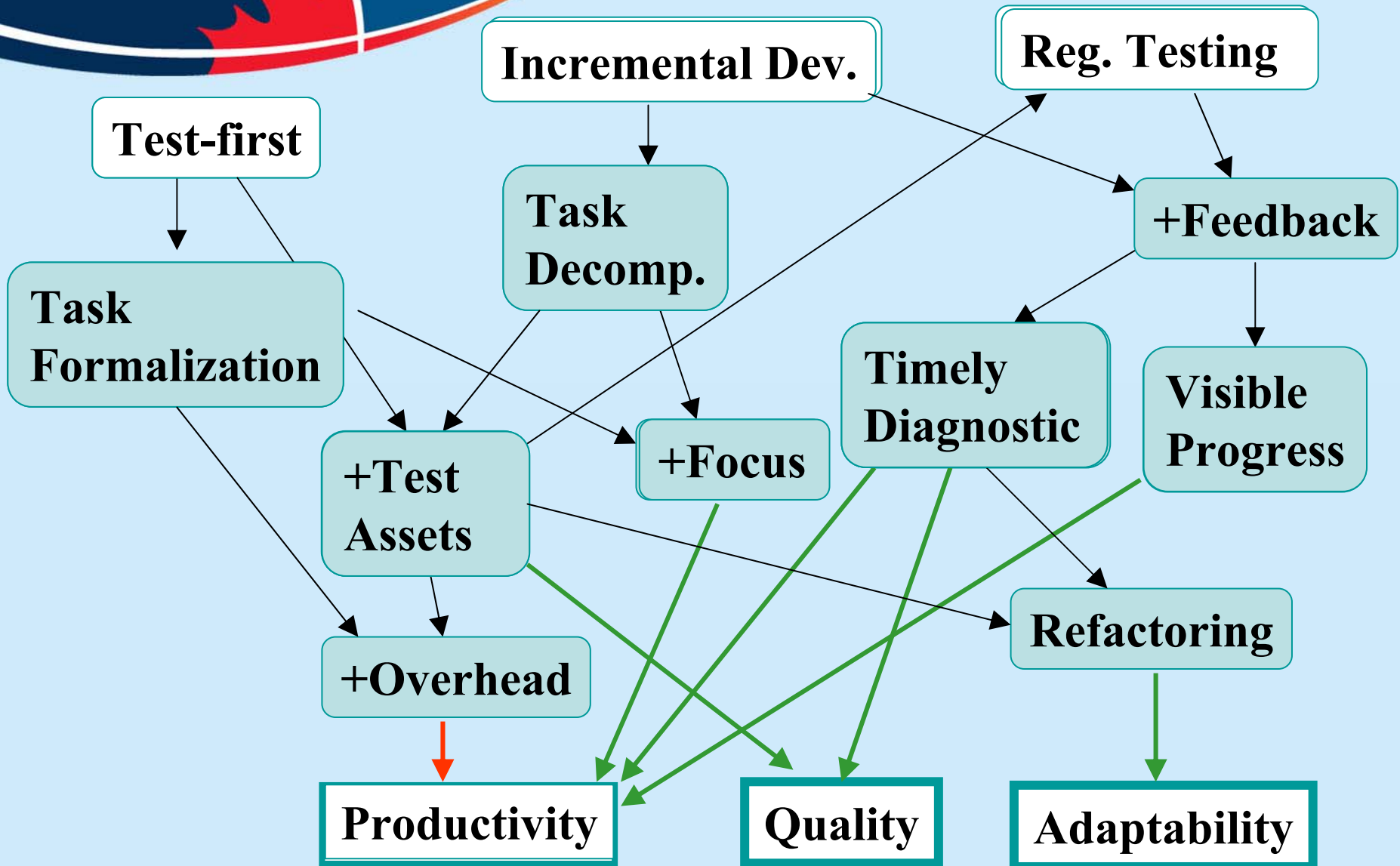


Iterative – TD

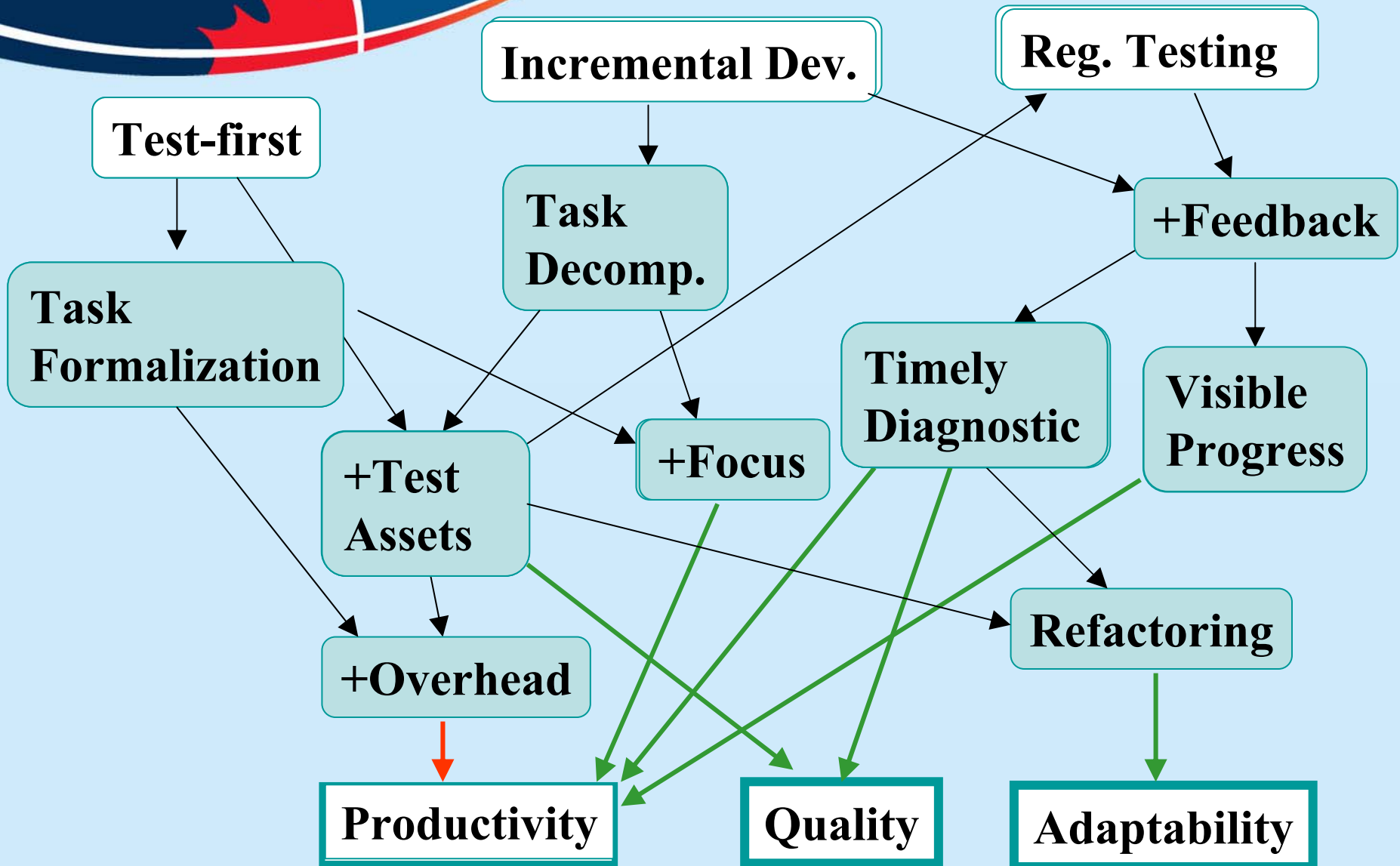


Pseudo-continuous – TD

A Theory of TDD



A Theory of TDD



Empirical results -- early

Authors Year	Type	Quality	Productivity	S/P
Müller+ 2001	CE	∅	∅	S
Williams+ 2003	CS	+	∅	P
George+ 2003	CE	+	-	P
Edwards 2003	CS	+	?	S
Long 2003	MA	+	?	P/S

CE: Controlled Experiment
CS: Case Study
MA: Meta-Analysis

S: Students
P: Professionals

Empirical results -- recent

Authors Year	Type	Quality	Productivity	S/P
Erdogmus+ 2005	CE	∅	+	S
Sanchez+ 2005	CS	+	-	P
Bhat+ 2006	CS+	+	-	P
Damm+ 2006	CS+	+	-	P
Confora+, 2006	CE	+	-	S
Flohr+ 2006	CE	?	+	S
Melnik+ 2005	CS+	+	NA	P

CE: Controlled Experiment
CS: Case Study
MA: Meta-Analysis

S: Students
P: Professionals

Empirical results – late breaking

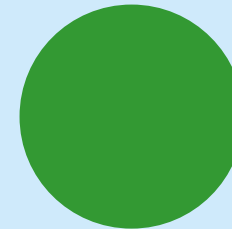
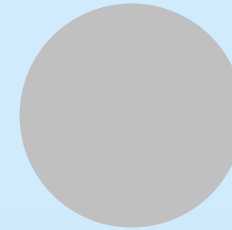
Authors Year	Type	Quality	Productivity	S/P
Sanchez+ 2007	CS	+	–	P
Siniaalto+ 2007	CS+	c+	?	P
Gupta+ 2007	CE	?	+	S
Janzen+ 2008	CE	i+ c+	NA	S, P
Madeyski 2007+	CE (n=1)	NA	+	S, P

CE: Controlled Experiment
CS: Case Study
MA: Meta-Analysis

S: Students
P: Professionals
i: internal quality
c: test coverage

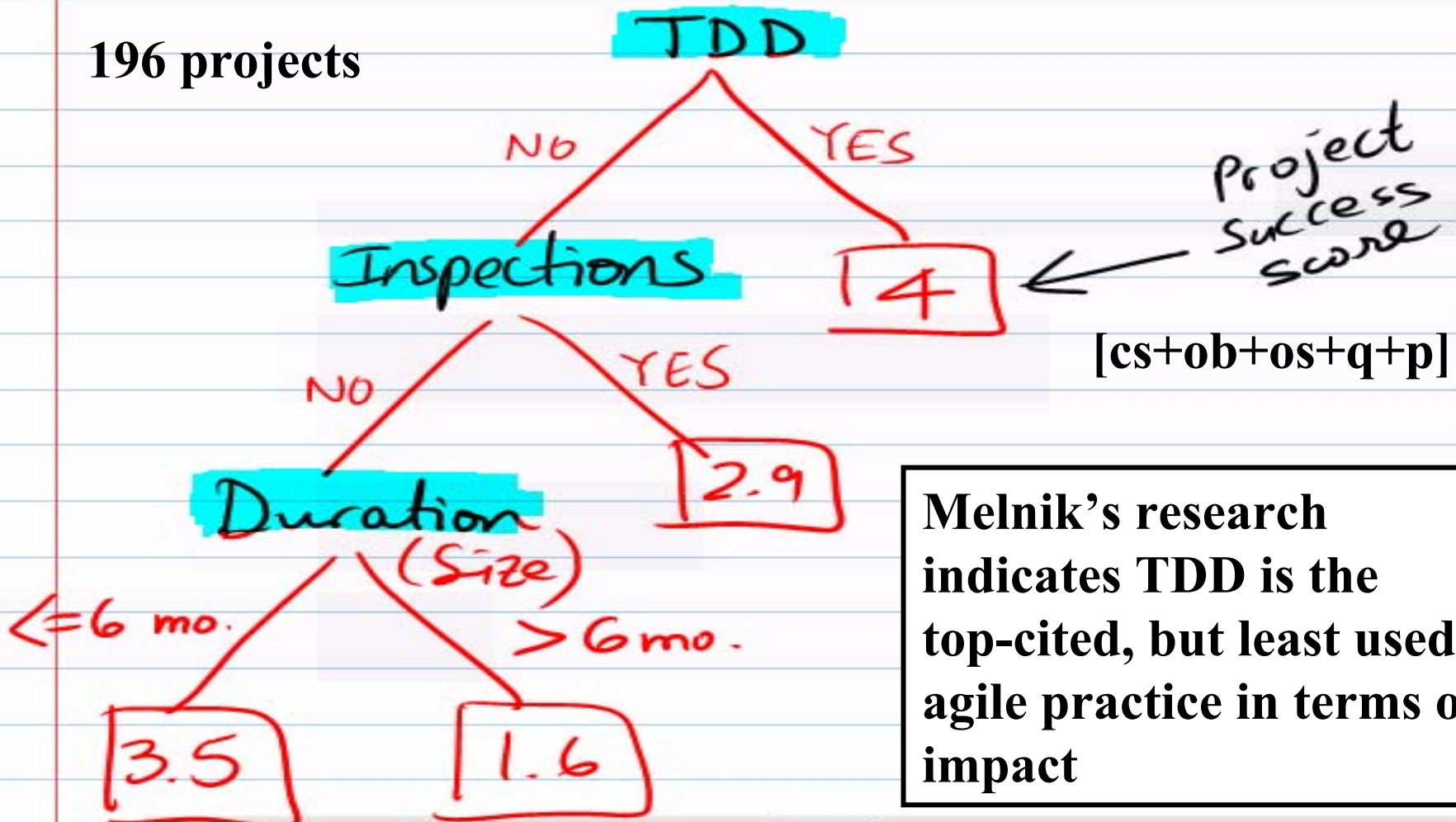
Summary of findings

- **Total no. of studies: 17**
- **Productivity**
 - improvement: 4
 - penalty: 5
 - no difference: 2
- **Quality (either internal or external)**
 - improvement: 12
 - penalty: 0
 - no difference: 2



Cutter Consortium Survey

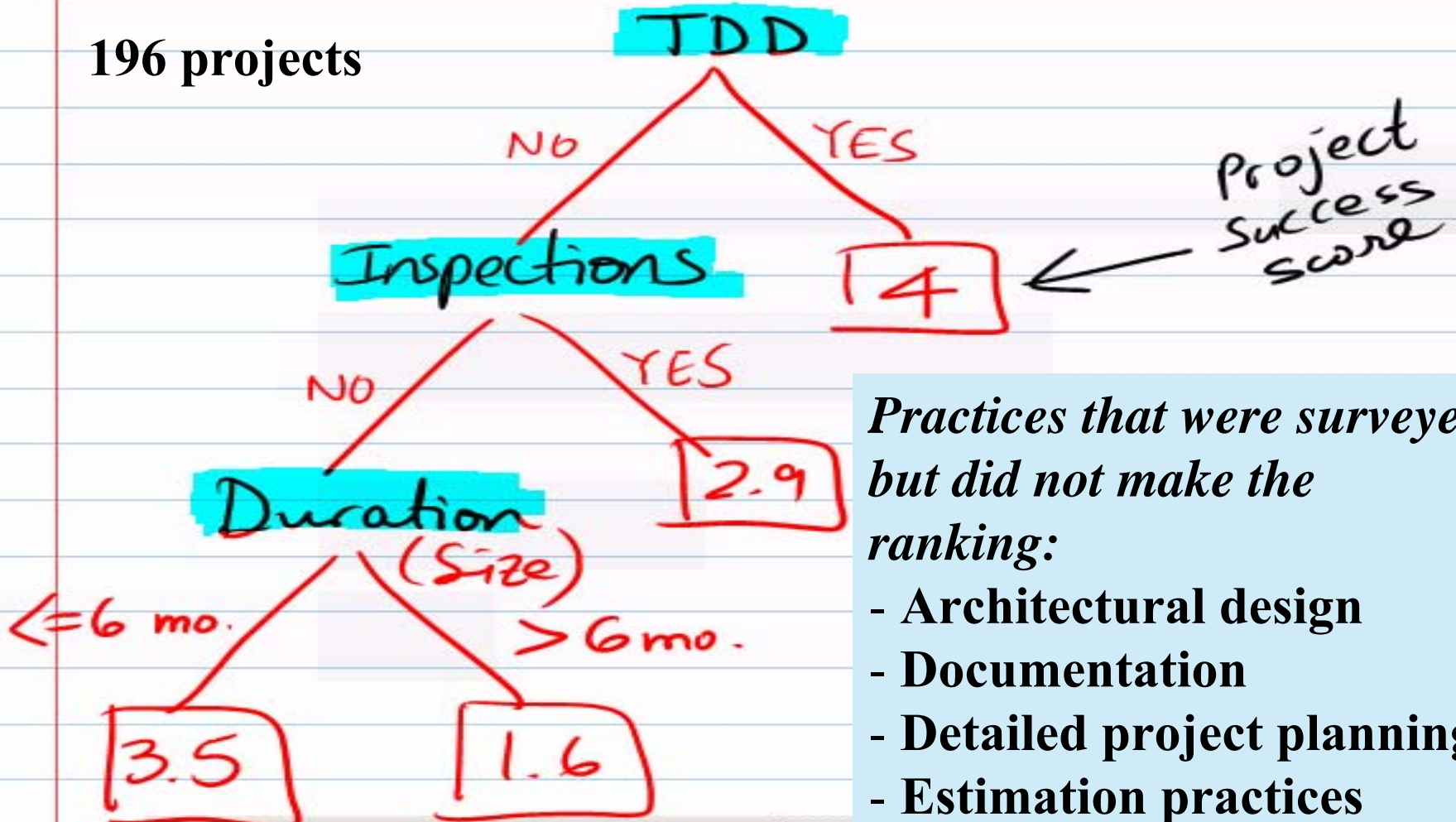
196 projects



Melnik's research indicates TDD is the top-cited, but least used, agile practice in terms of impact

Cutter Consortium Survey

196 projects



Practices that were surveyed but did not make the ranking:

- Architectural design
- Documentation
- Detailed project planning
- Estimation practices

Why is TDD controversial?

- **Misunderstood (myths?) -- “Blame the name”**
- **Incorporates & blends activities associated with traditionally distinct phases: *requirements, design, implementation, documentation, testing/QA***
 - Why should programmers write tests?
 - QA should be a separate function
- **Effectiveness unproven**
- **Too much work, too difficult to learn & apply**
- **Requires a certain way of reasoning and problem solving**
- **Weakest link**

TDD Myths

Myth

Truth

-
- Unit testing = TDD

- Testing \supseteq TDD

-
- It's a QA technique

- It's a development approach

-
- It's is a substitute to design & modeling, traditional QA

- It is orthogonal to design & modeling, traditional QA

-
- It's for junior/disorganized people
 - It's for advanced developers

- It requires skill and discipline, but junior developers can master it

-
- It curtails productivity

- Most who apply it consider it a productivity technique, but the jury is still out

General-purpose testing frameworks

Modules, Units

- **JUnit, TestNG – Java**
- **NUnit – C#, .NET**
- **CUnit – C**
- **CppUnit – C++**
- **PerlUnit – Perl**
- **PyUnit – Python**
- **RubyUnit – Ruby**
- **VBUnit – Visual Basic**
- **Sunit – SmallTalk**

General-purpose testing frameworks

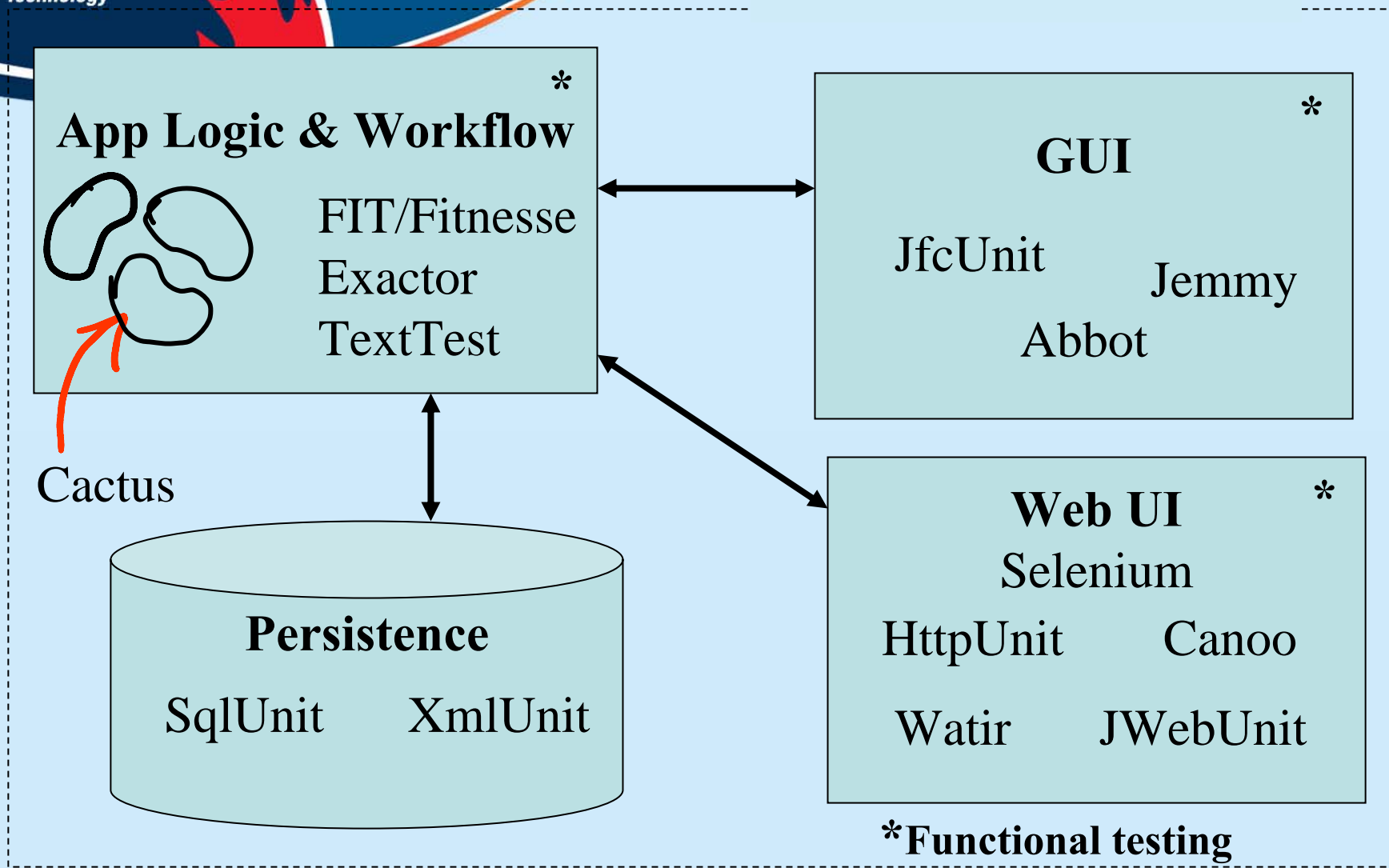
Modules, Units

- **JUnit, TestNG – Java**
- **NUnit – C#, .NET**
- **CUnit – C**
- **CppUnit – C++**
- **PerlUnit – Perl**
- **PyUnit – Python**
- **RubyUnit – Ruby**
- **VBUnit – Visual Basic**
- **Sunit – SmallTalk**

+Interactions, Integration

- **jMock, EasyMock – Java**
- **NMock – C#, .NET**
- **MockPP – C++**
- **Test::MockObject – Perl**
- **RSpec, Mocha – Ruby, Rails**
- **SMock – Smalltalk**

Specialized testing frameworks



Courtesy of Grigori Melnik

Performance: JMeter, JUnitPerf

Related paradigms

- **Acceptance Test-Driven Development (ATDD) – Fit/Fitnesse**
- **Story Test-Driven Development (STDD)**
- **Example-Driven Development**
- **Behavior-Driven Development (BDD) – RSpec, Rails**

TEST RESULTS

Assertions: 6 right, 0 wrong, 0 ignored, 0 exceptions

Set up: .EasyTixAcceptanceTests.Setup

fit.DoFixture	
start	ca.easytix.fixtures.FindBuyScenario

this will need to be externalized into setup

system acquires	1000	tickets for	Ottawa Senators
system acquires	10	tickets for	U2

user searches for event	U2
-------------------------	----

system displays search results

check	event	U2
check	tickets available	10

user buys	1	ticket with card	35295549283948592
-----------	---	------------------	-------------------

system returns eticket

check	number of seats purchased	1
check	event purchased	U2

Plus you need a modern IDE

- **Integration with testing framework**
- **Supports or works with continuous integration**
 - Version control support (Subversion, CVS)
 - Build support (Ant, CruiseControl)
- **Eclipse, IntelliJ/IDEA, Visual Studio, ...**
- **And a few extras...**

Conclusions

- **Compelling reasons to try out TDD**
- **Some evidence of its effectiveness (still not definitive about the productivity-quality trade-off)**
- **Levels at which it can be applied apply**
 - Personal: traditional TDD, unit testing
 - Team: acceptance/story/behavioral TDD
- **It may not work for everyone or for every project context**

Challenges

- **Application to:**
 - Legacy systems
 - Real-time & embedded systems
 - Highly distributed, decentralized systems, agents
- **Test-suite and execution management**
- **Adoption**



IEEE Software Special Issue on TDD – May/June 2007

Guest Editors' Introduction: TDD--The Art of Fearless Programming

**Ron Jeffries, Independent Consultant
Grigori Melnik, University of Calgary**

Professionalism and Test-Driven Development

Robert C. Martin, Object Mentor

Test-Driven Development of Relational Databases

Scott W. Ambler, IBM

Test-Driven Development of a PID Controller

Thomas Dohmke, Henrik Gollee, U. of Glasgow

Test-Driven GUI Development with TestNG and Abbot

Alex Ruiz, Yvonne Wang Price, Oracle

Envisioning Next-Generation Functional Testing Tools

Jennitta Andrea, clearStream

Incorporating Performance Testing in Test-Driven Development, Michael J. Johnson, IBM, Chih-Wei Ho, North Carolina State U., E. Michael Maximilien, IBM, Laurie Williams, North Carolina State U.

Learning Test-Driven Development by Counting Lines

Bas Vodde, Nokia Networks