XP and TDD
Extreme Programming and Test Driven Development

Andreas Leitner
Chair of Software Engineering
ETH Zurich

May 10, 2007
Goal

- Have overview of software development processes
- Understand what extreme programming is
- Understand what unit testing is
- Understand what test driven development is
- Differentiate the latter from test first development
Outline

Development Processes Overview

Extreme Programming

Unit Testing

Test Driven Development
Outline

Development Processes Overview

Extreme Programming

Unit Testing

Test Driven Development
Development Processes Overview

▶ Traditional Methods
  ▶ Waterfall model
  ▶ V model
  ▶ Spiral model
  ▶ Prototype model
  ▶ ...

▶ Agile Methods
  ▶ Extreme Programming
  ▶ Test Driven Development
  ▶ ...

Chair of Software Engineering

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich
Waterfall model
V model

- System Requirements
- Software Requirements
- Preliminary Design
- Detailed Design
- Implementation
- Acceptance Testing
- Integration Testing
- Unit Testing
- System Integration
Defect Cost

Relative cost to correct a defect

Spiral model

Figure from: Ghezzi, Jazayeri, Mandrioli, Software Engineering, 2nd edition, Prentice Hall
Project Management

- Programming competence varies greatly
  - 1:10 in a single group (Sackman, Erikson, Grant)
- Who introduces more bugs?
  - Experienced Developers
  - Beginners
Outline

Development Processes Overview

Extreme Programming

Unit Testing

Test Driven Development
XP: Motivation

- Schedule slips
- Project canceled
- Systems go sour
- Defect rate
- Doesn’t solve actual problem
- Business changes
- False feature rich
- Staff turnover
XP: Cost of Change

cost of change

time

cost of change

time

traditional

XP
XP: Rules

- The planning game
- Small Releases
- Metaphor
- Simple Design
- Testing
- Refactoring
- Pair programming
- Collective Ownership
- Continuous Integration
- 40h-Week
- On-Site Customer
- Coding Standards
XP: Programming in the Wild

- Is XP like “programming in the wild”?
Outline

Development Processes Overview

Extreme Programming

Unit Testing

Test Driven Development
Kinds of Testing

- Unit testing
- Integration testing
- System testing
- Acceptance testing
- Regression testing
Unit testing

- Tools
  - SUnit – Smaltoalk (first one)
  - JUnit – Java (www.junit.org)
  - cppunit – C++
  - PyUnit – Python
  - ...

JUnit: Example

```java
@Test public void simpleAdd() {
    Money m12CHF = new Money(12, "CHF");
    Money m14CHF = new Money(14, "CHF");

    Money expected = new Money(26, "CHF");
    Money result = m12CHF.add(m14CHF);

    assertTrue(expected.equals(result));
}

outcome: pass or fail
```
Good Test Cases

- Test Case passes, the software is good?
- Test Case fails, the software is bad?
- Revealing failure?
- Revealing potential failures?
- Satisfying coverage criteria?
Early one morning, a programmer asked the great master:

“I am ready to write some unit tests. What code coverage should I aim for?”

The great master replied:

“Don’t worry about coverage, just write some good tests.”

The programmer smiled, bowed, and left.
Later that day, a second programmer asked the same question. The great master pointed at a pot of boiling water and said:

“How many grains of rice should put in that pot?”
The programmer, looking puzzled, replied:

“How can I possibly tell you? It depends on how many people you need to feed, how hungry they are, what other food you are serving, how much rice you have available, and so on.”

“Exactly,” said the great master.

The second programmer smiled, bowed, and left.
Toward the end of the day, a third programmer came and asked the same question about code coverage.

“Eighty percent and no less!” Replied the master in a stern voice, pounding his fist on the table.

The third programmer smiled, bowed, and left.
After this last reply, a young apprentice approached the great master:

“Great master, today I overheard you answer the same question about code coverage with three different answers. Why?”

The great master stood up from his chair:

“Come get some fresh tea with me and let’s talk about it.”

After they filled their cups with smoking hot green tea, the great master began to answer:
Testivus On Test Coverage

“The first programmer is new and just getting started with testing. Right now he has a lot of code and no tests. He has a long way to go; focusing on code coverage at this time would be depressing and quite useless. He’s better off just getting used to writing and running some tests. He can worry about coverage later.”

“The second programmer, on the other hand, is quite experience both at programming and testing. When I replied by asking her how many grains of rice I should put in a pot, I helped her realize that the amount of testing necessary depends on a number of factors, and she knows those factors better than I do – it’s her code after all. There is no single, simple, answer, and she’s smart enough to handle the truth and work with that.”
“I see,” said the young apprentice, “but if there is no single simple answer, then why did you answer the third programmer ‘Eighty percent and no less’?”

The great master laughed so hard and loud that his belly, evidence that he drank more than just green tea, flopped up and down.

“The third programmer wants only simple answers – even when there are no simple answers ... and then does not follow them anyway.”

The young apprentice and the grizzled great master finished drinking their tea in contemplative silence.
Outline

Development Processes Overview

Extreme Programming

Unit Testing

Test Driven Development
TDD: Overview

- Evolutionary approach to development
- Combines
  - Test-first development
  - Refactoring
- Primarily a method of software design
  - Not just method of testing
TDD: The Process

1. Add a test
2. Run the tests
   - [Pass]
   - [Fail]
3. Make a little change
   - [Development continues]
4. Run the tests
   - [Fail]
   - [Development stops]
TDD = TFD + Refactoring

- Apply test-first development
- Refactor whenever you see fit (before next functional modification)
- Think for the moment:
  - Write new business code only when a test case fails
  - Eliminate any duplication you find
- Produce failure revealing test cases
TDD and Extreme Programming

- Easy to give in and skip some test cases
- Pair-programming can help
- Writing testable code helps
TDD: Consequences

- Incremental development
- Development environment must provide rapid response to small changes
- Components are designed highly cohesive, loosely coupled
- Developers learn to write good unit tests:
  - Run fast
  - Run in isolation
  - Use data that makes test case easy to read
  - Each test case is step towards overall goal
TDD & Documentation

- Programmers often do not read documentation
- Instead, they look for examples and play with them
- Good unit tests can serve as
  - Examples
  - Documentation
TDD: pros and cons

- **Pros**
  - Reduce gap between decision and feedback
  - Encourage developers to write code that is easily tested
  - Creates a thorough test bed

- **Drawbacks**
  - Time taken away from core development
  - Some code is difficult to test
References

▶ Andrew Hunt: The Pragmatic Programmer: from journeyman to master. Addison Wesley, 2000
▶ by Alberto Savoia: The Way of Testivus. JUnit Factory webpage