

CSE70: Lecture 9

- Projects are over
- Questions on the project?
- Questions on the final?
- Today
 - Final review

Software Development Process

- Defines
 - Phases, stages, methods, techniques, practices,...
- Why do you need a process?
 - Understanding, standardization, quality
 - Organization and support
 - Manage complexity, quality, and costs

Processes

- The Waterfall
 - Serial approach with Big Design Up Front (BDUF)
- Spiral model (Boehm 1988)
 - Iterative approach
- Incremental Development
 - No “*big-bang*” release
 - Sequence of *product baselines*

eXtreme Programming

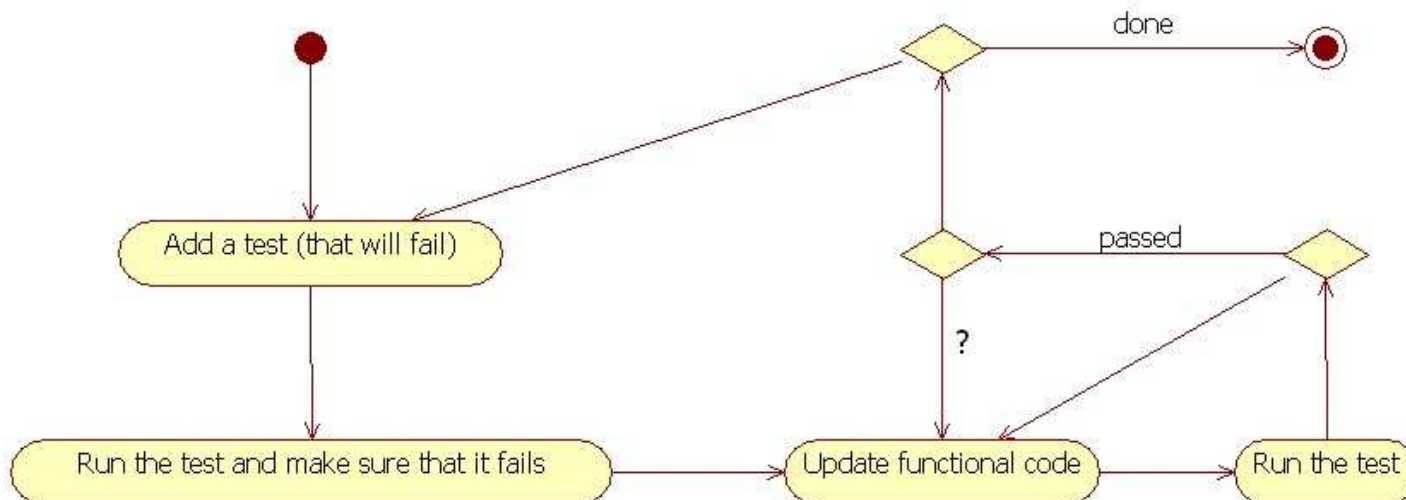
- 4 Values
 - Simplicity
 - Communication
 - Feedback
 - Courage
- 13 Practices
 - Whole Team
 - Planning Game, Small Releases, Customer Tests
 - Simple Design, Pair Programming, Test-Driven Development, Design Improvement
 - Continuous Integration, Collective Code Ownership, Coding Standard
 - Metaphor, Sustainable Pace

Aspects of Pair Programming

- Investigative paths (by Cockburn & Williams)
 - Economics
 - Satisfaction
 - Design quality
 - Continuous reviews
 - Problem solving
 - Learning
 - Team building and communication
 - Staff and project management

Test Driven Design

- Test Driven Design (TDD)
 - TFD + refactoring
- Not only testing
- 100% coverage



What is testing?

- “Program testing can be used to show the presence of bugs, but never to show their absence!” (Dijkstra)
- “Testing is a determined, systematic attempt to break a program that you think is working” (Kernighan & Pike)

Test (when writing the code)

- Boundary condition testing
- Test conditions (pre- and post-)
- Check error returns
- Test incrementally
- Verify conservation properties
- Measure test coverage

FLOOT

- Regression Testing
- QA
- Use Case Scenario Testing
- Prototype Walkthrough
- User Requirement Review
- Model Review and Walkthroughs
- User-Interface Testing (GUI Testing)
- Black-Box Testing
- White/Clear/Glass-Box Testing
- Boundary Value Testing
- Class Testing
- Class-Integration Testing
- Code Inspections
- Coverage and Path Testing
- Inheritance-Regression Testing
- Method Testing
- Function Testing
- Installation Testing
- Operations and Support Testing
- Stress Testing
- *Pilot* Testing
 - *Alpha, Beta*
- User Acceptance Testing (UAT)

Coverage

- Exercise statements with testing
 - Only 50-60% of the code exercised (Wiegers 2002)
- Statement coverage
- Branch Coverage
- Path Coverage
- Defined-use pairs
- Cyclomatic Complexity

Design

- UML
- OOD
 - Class=attributes + methods
 - Objects
 - Inheritance
 - Associations
 - Collaboration
 - Polymorphism
- Abstraction, Encapsulation, Information hiding
- Persistence
- Coupling and cohesion
- Components

Refactoring

- Change the code, but not the *observable* behavior
- Why, When, What

What about performance?

- Refactoring
- OOD
- Language (and its features)
- Compiler

Why don't we rewrite the Linux kernel in C++?

In fact, in Linux we did try C++ once already, back in 1992.

It sucks. Trust me - writing kernel code in C++ is a BLOODY STUPID IDEA.

The fact is, C++ compilers are not trustworthy. They were even worse in 1992, but some fundamental facts haven't changed:

- the whole C++ exception handling thing is fundamentally broken. It's especially broken for kernels.*
- any compiler or language that likes to hide things like memory allocations behind your back just isn't a good choice for a kernel.*
- you can write object-oriented code (useful for filesystems etc) in C, without the crap that is C++.*

In general, I'd say that anybody who designs his kernel modules for C++ is either

- (a) looking for problems*
- (b) a C++ bigot that can't see what he is writing is really just C anyway*
- (c) was given an assignment in CS class to do so.*

Feel free to make up (d).

Linus Torvalds, 2004

Pattern and Styles

- Why?
- Creational, Structural, Behavioral, Concurrency
- Architectural Styles
 - Layered architecture, Pipe-and-Filter
 - MVC, Event-Driven

The Cathedral and the Bazaar

- Cathedral
- Bazaar
- What about the users?
 - Source-aware
 - Co-developers

Tools

- Version Control
 - Subversion
 - bazaar
- Project Management
 - Trac
 - Bugzilla
 - Sourceforge
 - Maven
- Unit Testing
 - JUnit
 - Fixtures + validation + runners
 - cppUnit
 - More: www.opensourcetesting.org
- Mocking
 - JMock
 - Set up a mockery, mock objects, define expectations, execute test code

Tools

- Automation
 - Ant
 - Make
- Software measures
 - Metrics
 - Loc, ..., cyclomatic complexity
 - EcEmma
 - Coverage
- Acceptance Testing
 - concordion
 - Fit/Fitness
- Integration
 - Cruisecontrol
 - Maven
 - Continuum
- Integrated Development Environment
 - Integrates tools for software development

Questions?

- Jar executable
- Java Threads

CSE 70: Software Engineering

- What is Software Engineering?
 - The IEEE Computer Society defines software engineering as:
 1. The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software.
 2. The study of approaches as in (1).

"IEEE Standard Glossary of Software Engineering Terminology," [IEEE](#) std 610.12-1990, 1990, quoted at the beginning of Chapter 1:
Introduction to the guide

- Objectives
 - Developing non-trivial software systems requires a systematic approach and the use of adequate tools. The goal of this class is to help you understand the critical issues that arise in planning, design, integration, testing, and team coordination in software development.

The End

- Final
 - Saturday
 - 11:30a - 2:29p
 - EBU3B 2154