EECE 310

Agile Process Models: Extreme Programming

Groups

- Groups are created.
  - See Connect for the groups list
  - Groups are created on **GitHub** and **Connect**
  - Get to know your team mates

First lab assignment

- Lab sessions start next week
  - Attendance is mandatory
  - See Connect for further instructions

Presentation Topics

- Each group will present once
- Choose 3 topics, submit by Monday
- A list of possible topics is provided
  - But other ideas are welcome.

- Each group has 15 minutes (slides + Q&A)
- Each session 3 groups

Next Friday

- Groups 1, 2, 3
- Benefit: you get to choose your first choice of topic.

Activity

- Find in which group you are
- Find your group members
- Discuss 3 potential topics for your group's presentation

- **Groups 1, 2, 3** write down your choice and hand it in.

Reading material!

- What is eXtreme Programming? ***(required)***
  - http://www.xprogramming.com/xpmag/whatisxp.htm
- Scrum by Michael James ***(required)***
  - **Connect (under reading material)**

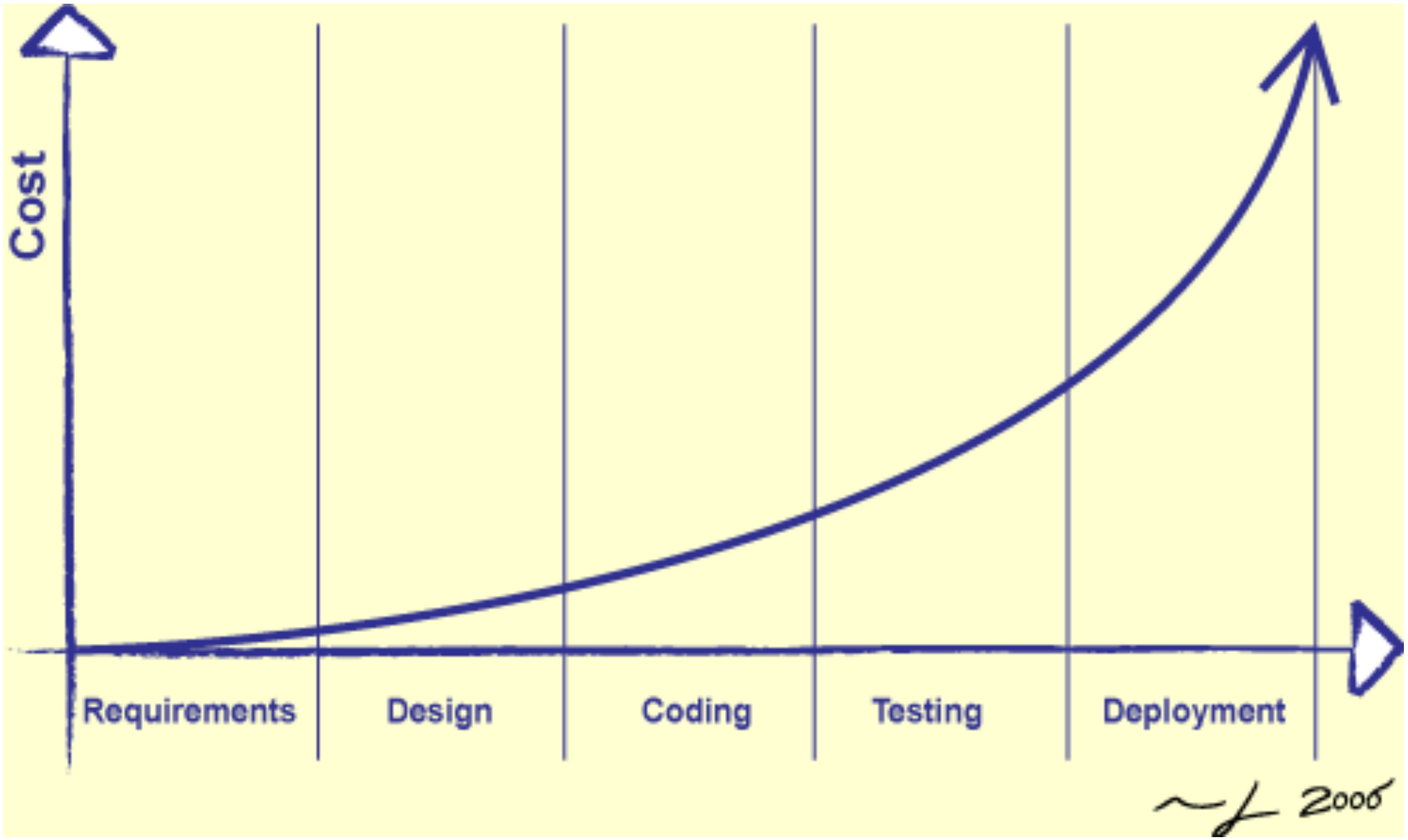
See Connect -> Reading Material

Software Process Model: Waterfall

- What was that again?

Drawbacks of Waterfall?

The Boehm Curve



Source: Greg Wilson, http://www.cdf.toronto.edu/~csc301h/fall/lec/01-intro.html

Agile manifesto

In 2001, 17 software developers came together to discuss lightweight development methods, which resulted in 4 basic principles.

Agile manifesto

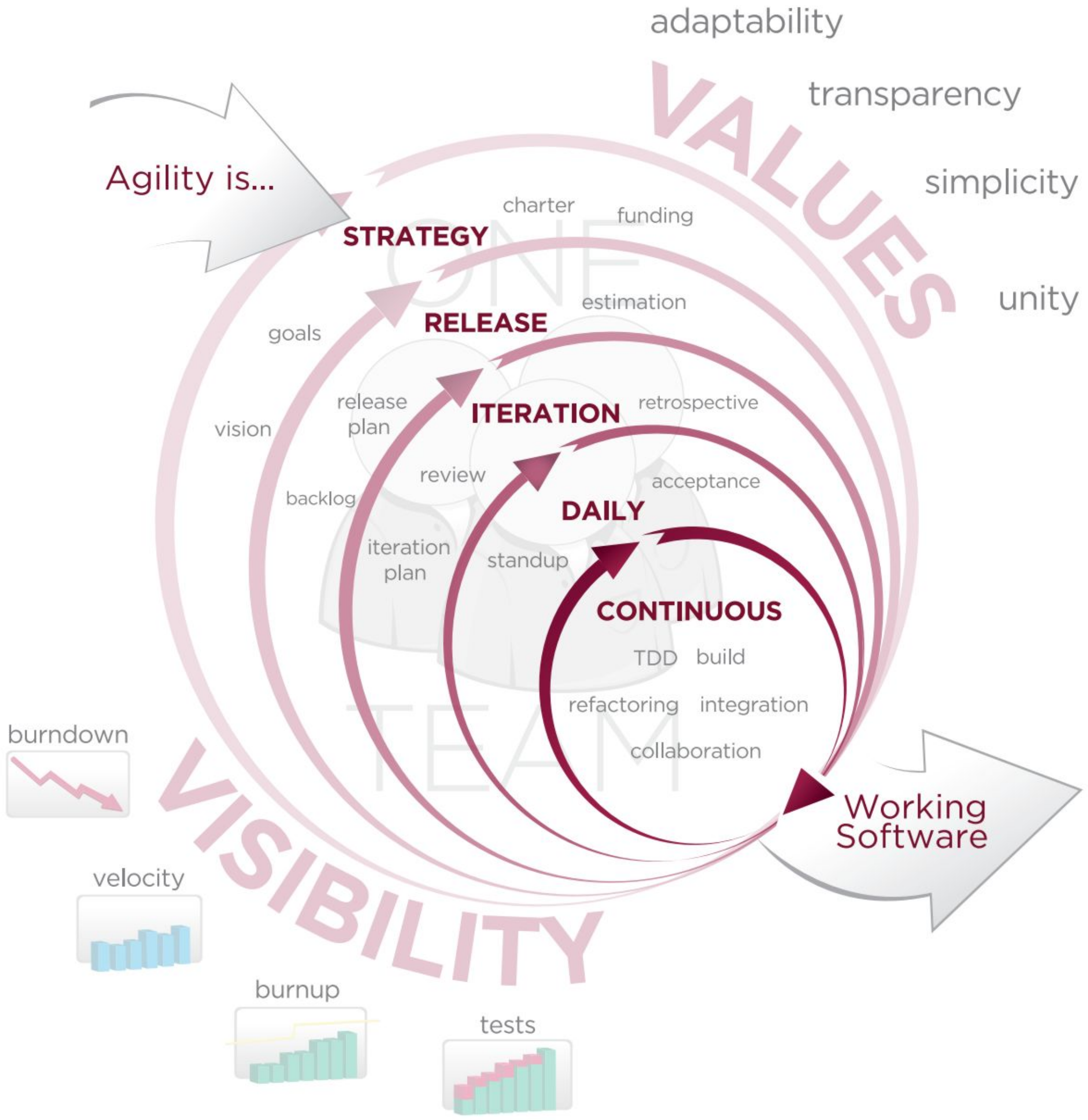
- **Individuals and interactions** over *processes and tools*
- **Working software** over *comprehensive documentation*
- **Customer collaboration** over *contract negotiation*
- **Responding to change** over *following a plan*

What is “agility”?

- Adapt to change
  - Particularly changes in requirements
  - Use frequent, short iterations to **flatten cost curve**

- A sustainable process
  - Find a method that **delivers reliably**

AGILE DEVELOPMENT



ACCELERATE DELIVERY

In-class activity

Often, the customer does not have a complete understanding of what she actually wants (requirements) at the beginning of a project.

What solution to this issue does the Agile approach suggest ?

A. Discuss the problem in detail with the customer at the beginning of the project.

B. Significantly larger group of developers at the beginning of the project.

C. Begin writing code and the requirements will emerge.

D. Use short development cycles to allow for incremental requirements development.

E. Write requirements in the customer's language and discuss.

In-class activity

Often, the customer does not have a complete understanding of what she actually wants (requirements) at the beginning of a project.

What solution to this issue does the Agile approach suggest ?

A. Discuss the problem in detail with the customer at the beginning of the project.

B. Significantly larger group of developers at the beginning of the project.

C. Begin writing code and the requirements will emerge.

D. Use short development cycles to allow for incremental requirements development.

E. Write requirements in the customer's language and discuss.

Agile Methods

- One Agile Methodology (principles)
- Many Agile Methods (instantiations)

- Anyone used an agile approach?

Agile Methods

- Extreme Programming (XP)
  - Specific practices - customer driven development, small teams, daily builds
  - Use XP for design, develop and test
- Scrum
  - Project management approach, relying on self-organizing independent teams
  - Use Scrum for managing your software project
- Several others – Crystal, FDD, DSDM

eXtreme Programming (XP)

- Developed by Kent Beck in mid-90s
- Popular agile method
  - but not necessarily always successful

Values of XP

Five principal values:

1. **Communication**: common metaphors, frequent verbal communication, customer involvement
2. **Simplicity**: do the simplest thing that could possibly work, then refactor
3. **Feedback**: from the code (unit tests), the customer (co-location), the team (planning game)
4. **Courage**: be willing to throw things away
5. **Respect**: don't do things that make work for others more difficult (e.g. commits that break the build)

Agile Tools & Techniques (XP)

- User stories
- Unit Testing (xUnit)
- Test-driven development
- Continuous integration (eg. Jenkins)
- Pair programming
- Refactoring

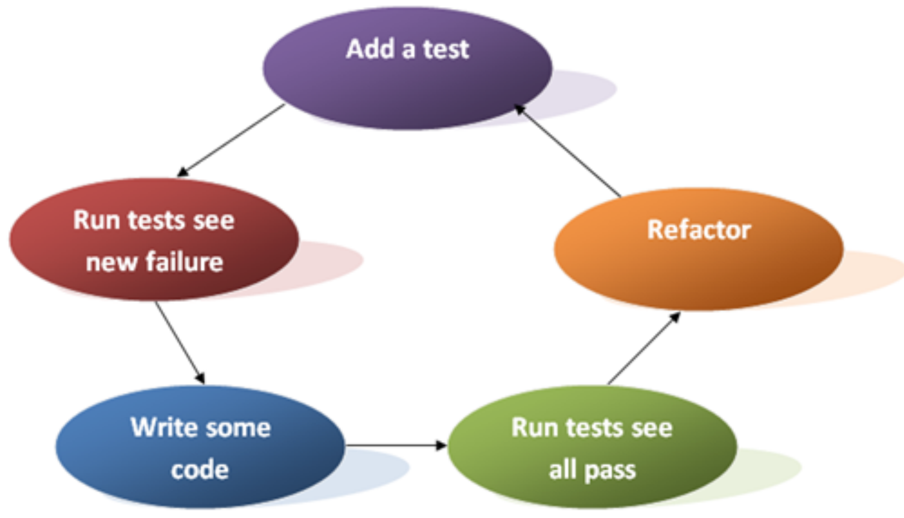
User stories

- A ~3 sentence description of what the system should do (informal specs)
- Written in the customer's language, from the customer's point of view
- Only enough detail to make a **low-risk estimate** of how long it would take to implement (1, 2 or 3 week estimate in "ideal development time")
- Typically takes Role-Goal-Benefit form:
  - "As a <ROLE>, I want to <GOAL> in order to <BENEFIT>"
  - As a student, I need to log on to Connect in order to download the assignment.

The (J)Unit Testing Framework

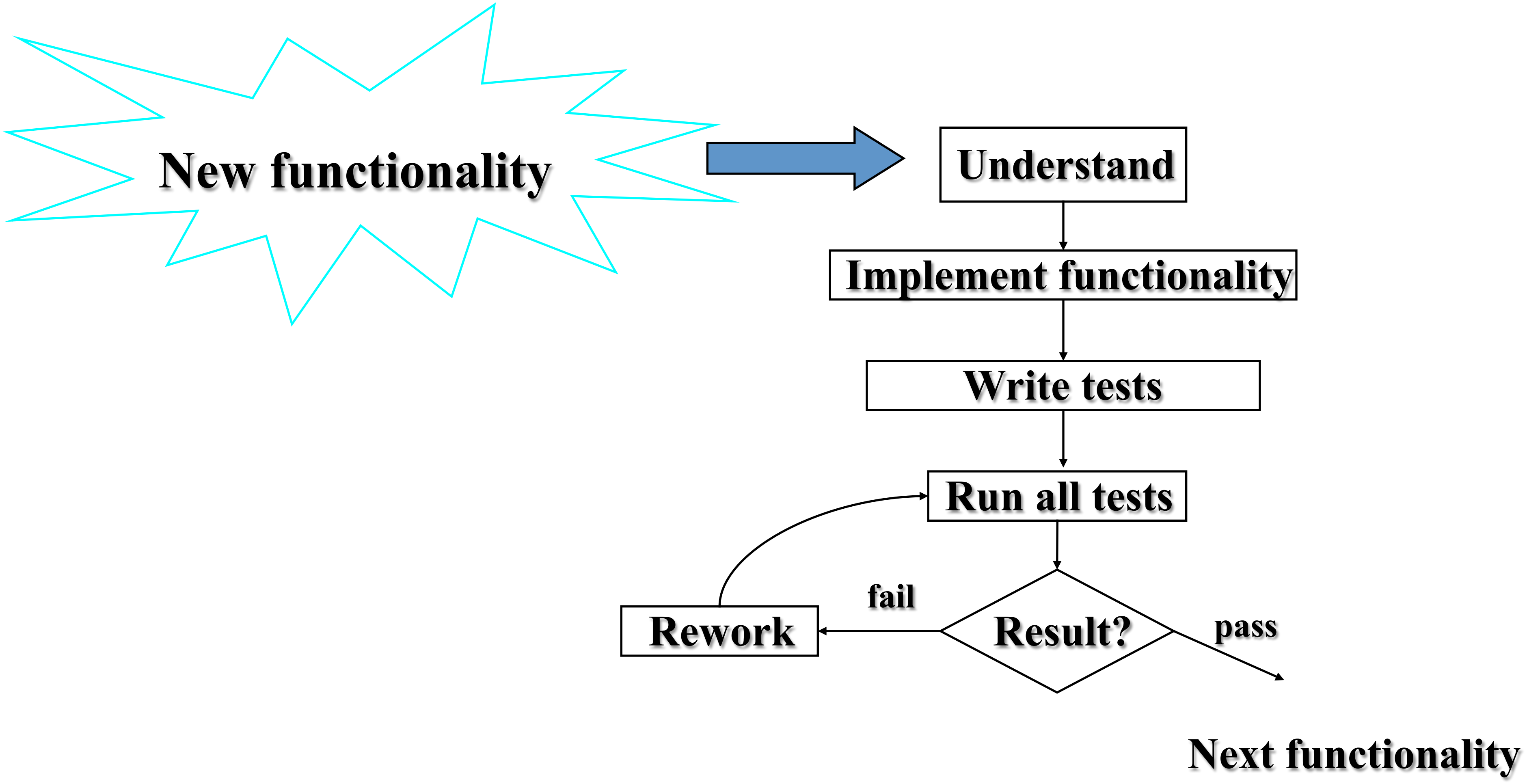
- Test one small unit of code at a time
  - Examples: methods, functions, classes
- Automatically verify results of that unit
  - Inputs, call, expected output
  - Example: inputs: {2, 3}, call: result = calculator.add(2, 3), expected output: 5
    - Check: (result == 5)?
- Organize unit tests into test suites
- Light-weight & easy to learn
- Typical process used:
  - "Test-Driven (test-first) development"

Test-Driven Design

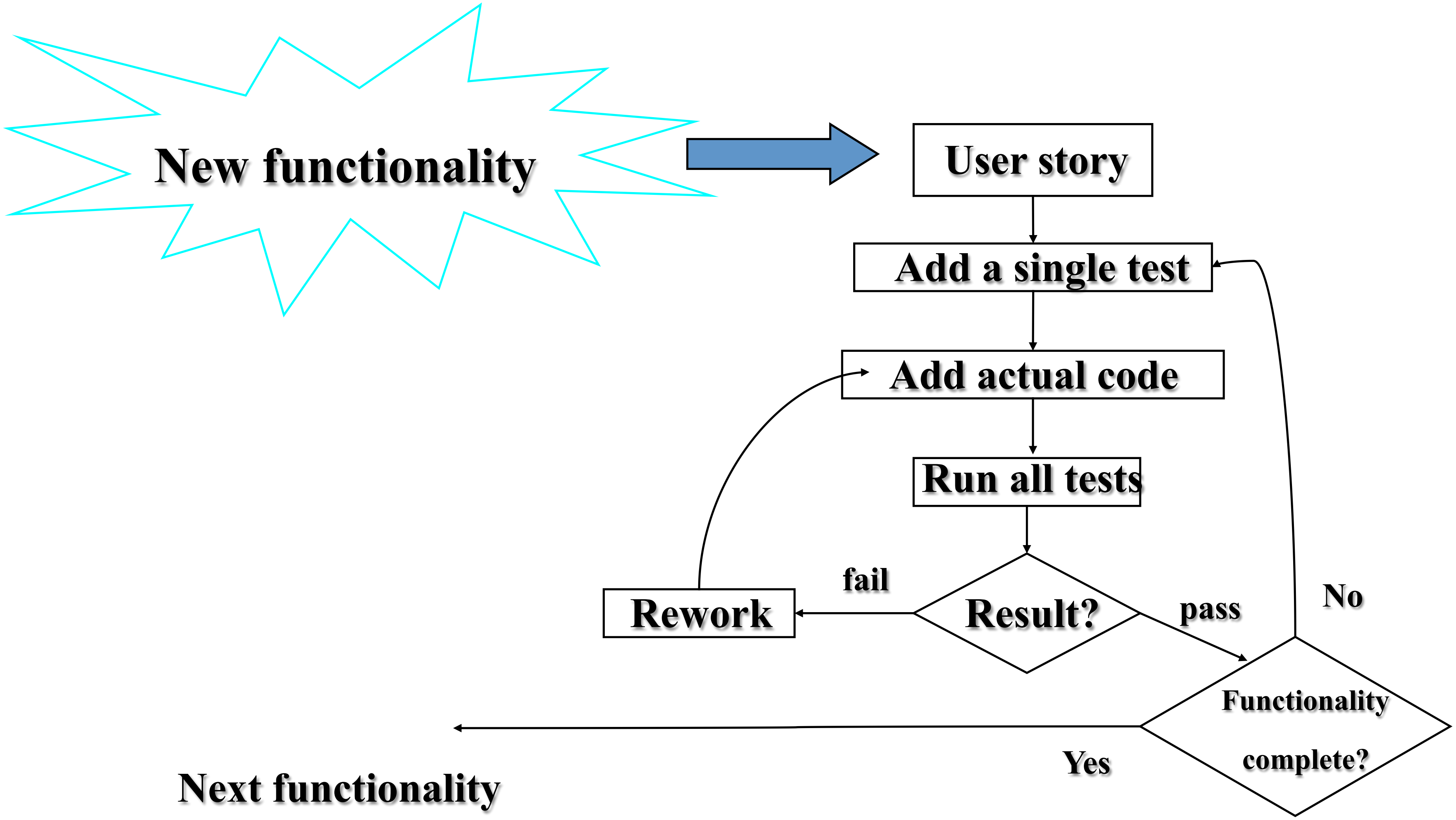


1. Write "user story"
2. Turn into set of testable scenario's
3. Implement one scenario as (failing) automated test case
4. Implement the underlying feature
5. Ensure the test passes
6. Refactor code and test cases
7. Repeat for next scenario

Classical Approach: Test last



Test-Driven Development: Test first



3 Rules of TDD

1. You are not allowed to write any production code unless it is to **make a failing unit test pass.**
2. You are not allowed to write **any more of a unit test than is sufficient to fail**.
3. You are not allowed to write **any more** production code than is sufficient **to pass the one failing unit test**.

Best Unit Testing Practices

- **During Development:** When you need to add new functionality to the system, write the tests first. Then, you will be done developing when all the tests run.

- **During Debugging:** When someone discovers a bug in your code, first write a test that will succeed if the code is working (expected behavior). Then debug/repair until the test succeeds.

Discuss: pros and cons of Test-driven Development

Advantages of Test-Driven

- Tests define small units of work
- Tests document expected functionality
- Tests are contracts for methods
- Later additions can be "regression" tested
  - Failure indicates breakage
- Think (at least) twice for each functionality to be added

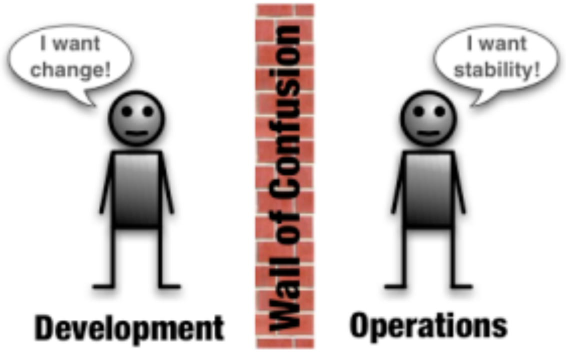
Drawbacks of Test-Driven

- Designing good tests is difficult
- Can require extra implementation time
- Hard to do for real systems
- GUIs can be hard to test
- Passing tests can give you false assurances about the quality of you code

Continuous Integration: The Daily Build

- Any modification may corrupt build
  - Compilation problems, tests don't run, ...
- Ensure *clean build* every **night/hour/minute**
  - Clean check out from version control system
  - Execute: mvn clean compile test deploy
- ***Continuous Integration****:*
  - Key practice of *extreme programming,* various *agile methods,* Microsoft, IBM, Apache, ...
  - Cruise control, Jenkins, Travic CI

Continuous Delivery

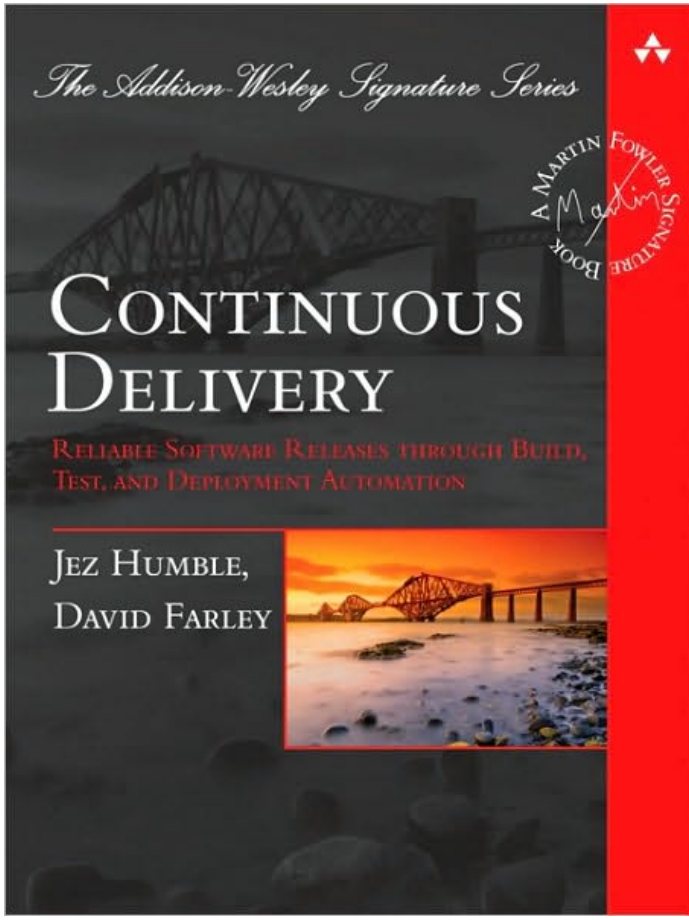


- Relatively new phenomenon (DevOps)
- Used mainly in web-based projects
- Multiple releases per day
  - 3, 10, 100, …

- Pros: Fast releases, user feedback, newest features
- Cons: Security vulnerabilities, bugs

Continuous Delivery

- Explains how to design/ set up your continuous delivery process
- Including CI, deployments, etc.
- Explains good software development practices that are necessary for CD


The Addison-Wesley Signature Series
A Martin Fowler Signature Book
Continuous Delivery
Reliable Software Releases through Build, Test, and Deployment Automation
Jez Humble, David Farley

Agile Tools & Techniques (XP)

- **User stories**
- **Unit Testing (xUnit)**
- **Test-driven development**
- **Continuous integration (eg. Jenkins)**
- Pair programming
- Refactoring

Submit Presentation Topics

- Missing 14 groups
- Deadline is tonight!
- Submit via Connect (not email, not piazza)
- We have too many web/mobile already. Choose something different

What Is Pair Programming?

Pair Programming

- Two programmers work side-by-side at one computer
  - One writes code, the other checks (for mistakes)
- Continuously collaborating on the same design, algorithm, code, and test
- Goal: produce a higher quality of code than that produced by the summation of their solitary efforts

Expected Benefits of Pair Programming

- Higher **product quality**
- Improved cycle time
- Enhanced learning
- **Pair rotation**
  - Ease **staff training** and transition
  - Knowledge transfer
  - Enhanced team building

- Increased programmer satisfaction!(?)

Issues in Pair Programming?

Issues: Partner Picking Principles

Expert paired with an Expert

Expert paired with a Novice

Novices paired together

Professional Driver Problem

Culture, beliefs

Refactoring

- Refactoring is:
  - restructuring (rearranging) code...
  - ...in a series of small**, semantics-preserving**
  - ...in order to make the code easier to **understand**, **maintain** and **modify**
- Refactoring is *not* just any old restructuring
  - You need to **keep the code working**
  - You need small steps that **preserve semantics**
- There are numerous well-known refactoring techniques
  - You should be at least somewhat familiar with these before inventing your own

When to refactor

- You should refactor:
  - Any time that you see a better way to do things
    - "Better" means making the code easier to understand and to modify in the future
  - Any time you detect "bad smells" in the code
- You should *not* refactor code that:
  - Does not have unit tests
  - Is being deployed right after the refactoring
  - Is already clean and maintainable

What is that smell in your code?

Examples of bad smells include:

- Duplicate Code
- Unused Code
- Long Methods
- Large Classes
- Long Parameter Lists
- Cross-cutting concerns
- Unreadable class/method/variable names (eg. method x345yyg)

Class Activity – Scenarios

- Discuss in your group:
  - **What are some of the challenges of applying agile in practice?**

Write it down, write your names, and hand it in!

Drawbacks of Agile

- Harder to enforce with inexperienced programmers
- Requires close customer involvement
- Increases the risk of feature creep
  - Adapting is good, but you need to draw the line
- Can be inefficient
  - It saves time, but
  - too much refactoring/change can be costly

Agile (XP) Tools & Techniques (revisited)

- User stories
- Unit Testing (xUnit)
- Test-driven development
- Continuous integration
- Small and frequent releases (Not XP, but CD in DevOps)
- Pair programming
- Refactoring