XP

extreme programming

Lecture 8

XP

- EXtreme Programming explained: embrace change
  by Kent Beck, Addison Wesley, September 1999.

- What is XP ?
  - a light-weight methodology for small to medium sized teams developing software in the face of vague and rapidly changing requirements
- Why XP ?
  - traditional software development methodologies are too rigid and over-constraining,
  - they do not work for small projects,
  - encourage "if there's a problem throw more people at it" thinking.
- Is XP a religion ?

XP

- **XP in context**
  - Kent Beck has a Smalltalk background
  - He is part of the Design Pattern community
  - He has worked with Fowler on Refactoring

- **XP and traditional Methodologies**
  - XP runs counter to almost all software engineering practice,
  - XP is not a solution for all problems (mostly for small teams),
  - XP is programmer friendly

Why XP ?

- **The Risks of Software Development**

  - Schedule slips — the delivery date is always six months in the future

  - Project canceled — after many slips, project canned

  - System goes sour — after a couple of years of operation and some changes, bugs start to appear

  - Defect rate — so buggy that it is not used

  - Business misunderstood —software does not answer all the right questions

  - Business changes — the system answers the wrong (out of date) questions

  - False feature rich — lots of unused features

  - Staff turnover — where have all the good programmers gone?

Why XP ?

- **XP to the rescue**

  - Schedule slips
    - short release cycles to limit the scope of slips
    - within release XP uses 1 to 4 wks customer-requested feature iterations
    - within an iteration, 1-3 day tasks
    - implement most important features first, to minimize the impact of slips
  - Project canceled
    - Customer involvement to choose the smallest possible release, to minimize potential bottlenecks and maximize software value.
  - System goes sour
    - create and maintain a comprehensive suite of tests
    - run tests after every change
  - Defect rate
    - unit test (programmer defined)
    - functional tests (user defined)

Why XP ?

- **XP to the rescue**

  - Business misunderstood
    - customer is an integral part of the team
    - specification continuously refined
  - Business changes
    - shorter release cycles imply less change during development
    - unimplemented features can be replaced at no cost
  - False feature rich
    - only highest priority tasks are addressed
  - Staff turnover
    - religion

The XP premise

- The cost of change plays a key role in most software engineering methodologies

Cost of change
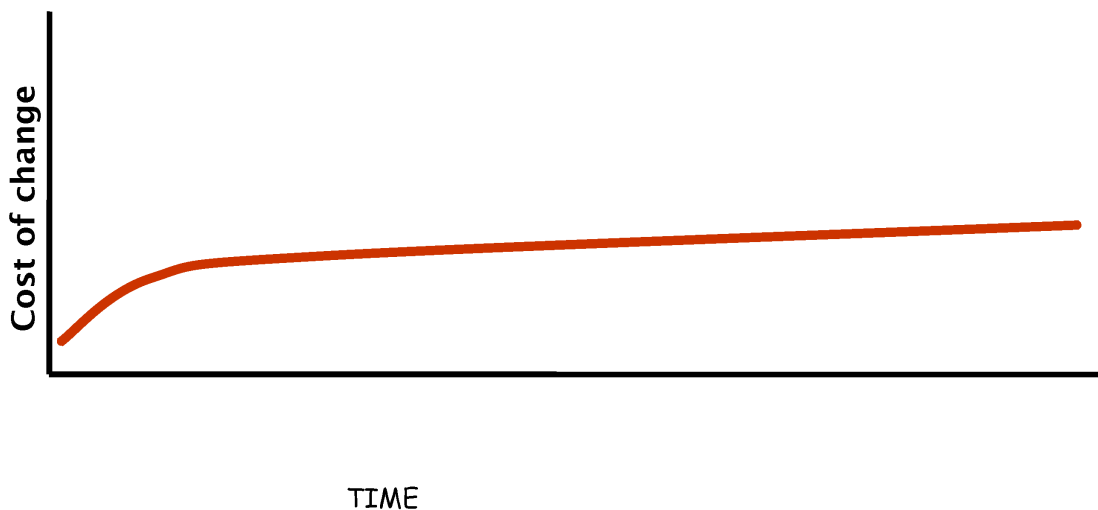
REQUIREMENTS   ANALYSIS DESIGN   IMPLEMENTATION TESTING PRODUCTION
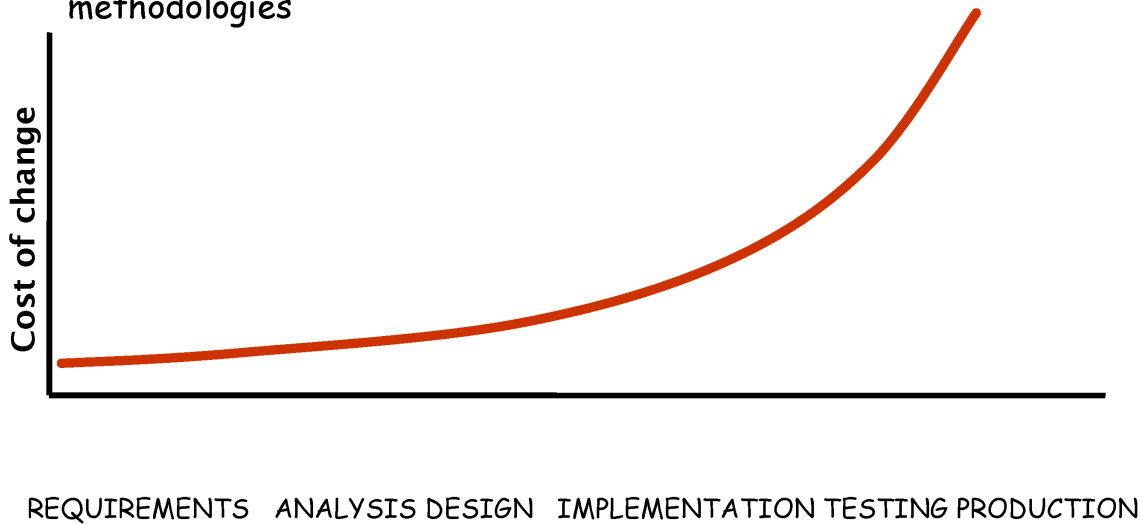
The XP premise

- What if the cost of change did not rise exponentially?

Cost of change

TIME

The XP premise

● How can the cost of change be contained ?

    ● Objects (and dynamic binding)
    ● Object databases (for persistence)

    ● but also
      ■ Simple designs with no unused generality
      ■ Automated tests to catch any accidental behavioral changes
      ■ Experience in modifying designs

The XP premise

● How does the premise affect software development?

Instead of making big decisions early and little ones late,

XP makes each decision quickly, backs them with tests,

and is also prepared to modify designs.

The Basics

- XP relies on 12 principles that are used as guides during the development process.

- XP separates software development into 4 activities, these are roles a software engineer can play

- XP advocates 12 practices that describe how to approach the development process

The 12 XP principles

1. Rapid feedback
2. Assume simplicity
3. Incremental change
4. Embracing change
5. Quality work
6. Small initial investment
7. Concrete experiments
8. Open, honest communication
9. Accepted responsibility
10. Local adaptation
11. Travel light
12. Honest measurements

The 12 XP principles

1. Rapid feedback
   * generate feedback, interpret it and put experience in the system as frequently as possible
   * business learns the benefits and shortcoming of the systems
   * programmers lean how to best test, design, implement
   * seconds/minutes instead of weeks/months

2. Assume simplicity
   * do not design for reuse
   * plan for today and trust your ability to add complexity in the future

The 12 XP principles

3. Incremental change
   * designs change a little at a time
   * plans change a little at a time
   * teams change a little at a time

4. Embracing change
   * best strategies preserve most options while solving the pressing problems

The 12 XP principles

5. Quality of work
   - quality is not a free variable: the only possible values are "excellent" and "insanely excellent"

6. Small initial investment
   - tight budgets force programmers and customers to focus on essentials
   - avoid comfort

7. Concrete experiments
   - every abstract decision should be tested
   - the result of a design session should be a series of experiements

The 12 XP principles

8. Open, honest communication
   - deliver the bad news early

9. Accepted responsibility
   - responsibilities should not be given, they should be accepted

10. Local adaptation
    - there are no fixed rules

11. Travel light
    - keep things small, maintain only the essential

11. Honest measurements
    - strive for accurate measurement of productivity

The 4 XP activities

🔴 CODING

- ⬥ coding as learning

- ⬥ coding as communication

- ⬥ code as end result

- ⬥ code as specification

🔴 TESTING

- ⬥ anything that can not be measured does not exist

- ⬥ without test, software is useless

- ⬥ tests are not only for functional requirements they are also for performance and adherence to standards

- ⬥ "test infected" -- do not code before having tests

- ⬥ write only tests that could possibly fail (but beware about that possibly)

- ⬥ test keep the program alive longer

- ⬥ testing improves productivity

The 4 XP activities

🔴 LISTENING

 ⬥ listening to customers

 ⬥ find rules that encourage useful communication

 ⬥ find rules that discourage useless communication

The 4 XP activities

🔴 DESIGNING

 ⬥ organize the logic of the system

 ⬥ good design ensures that every piece of logic has only one home

 ⬥ good design allows the extension of the system with changes in only one place

 ⬥ bad design is seen when one modification requires many changes

 ⬥ complexity is a source of bad design

 ⬥ design is a daily activity of all programmers

The 12 XP practices

1. The Planning Game — quickly determine scope of next release
2. Small releases — put a simple system in production quickly then release new version on a short cycle
3. Metaphor — guide development with a simple shared story
4. Simple design — system should be as simple as possible, complexity should be removed if at all possible
5. Testing — continually write unit tests, customers write functional tests
6. Refactoring — restructure the system without changing behavior
7. Pair programming — all code written with 2 programmer at 1 machine
8. Collective ownership — anyone can change code anywhere anytime
9. Continuous integration — integrate and build many times a day
10. 40-hour week — work no more than 40h/wk as a rule
11. On-site customer — include a real, live user on the team full time
12. Coding standards — code in accord. to rules emphasizing communication

January 02, 2009   468

The 12 XP practices

- The Planning Game
  - Business people decide about
    - Scope -- how much of the problem must be solved to have a valuable product
    - Priority -- which features take precedence
    - Composition of releases -- how much do we put in each release
    - Dates of releases -- what the dates at which we need to show something
  - Technical people decide about
    - Estimates -- how long will a feature take to implement
    - Consequences -- choices of the software environment and their impact on the product
    - Process --how will the team be organized
    - Detailed scheduling -- which are the tasks that are the riskiest, move them first, accommodate business

January 02, 2009   469

The 12 XP practices

● Metaphor

  ◦ every project is guided by a single overarching metaphor

  ◦ vocabulary should be consistent with the metaphor

  ◦ give a coherent story within which to work, a story that can be easily shared by business and technical

  ◦ a metaphor is a system architecture that is easy to communicate

The 12 XP practices

● Simple design

  ◦ The right design ins one that:
    ▪ (1) runs all tests
    ▪ (2) has no duplication
    ▪ (3) states every intention
    ▪ (4) has the fewer classes and methods

The 12 XP practices

● Testing

  ◆ any feature without an automated test does not exist

  ◆ programmers write unit tests

  ◆ customers write functional tests

  ◆ write test only for method that could possibly break

The 12 XP practices

● Refactoring

  ◆ before changing the program: Is there a way of modifying the program to make adding this new feature easier?

  ◆ after changing the program: Is there a way to make the program simpler?

  ◆ you refactor only when the systems requires you to

The 12 XP practices

● Pair programming

- all production code is written with two people looking at one machine
- there are two roles in each pair:
  - one partner is thinking about implementation
  - the other is thinking strategically
    - is this whole approach going to work
    - what test cases may fail
    - can we simplify the system to make this problem go away
- pair programming is dynamic, different pairs each time
- pair programming spreads knowledge

The 12 XP practices

● Collective ownership

- anybody who sees an opportunity to add value to any portion of the code is required to do so at any time
- chaos is adverted by testing

The 12 XP practices

Object Oriented Software Engineering

- Continuous integration
  - code is integrated and tested several times a day
  - integration ends when 100% of tests are passed

The 12 XP practices

Object Oriented Software Engineering

- 40 hour weeks
  - be fresh and rested
  - overtime is a symptom of serious problems on the project

The 12 XP practices

● On site customer
  ◦ real customers are need full time
  ◦ provide instant feedback
  ◦ keep development on track

The 12 XP practices

● Coding standards
  ◦ the standard is indispensable
  ◦ it should not be possible to tell who wrote a piece of code
  ◦ the standard must be accepted by the whole team

The 12 XP practices

On-Site Customer

Planning Game

40 hour week

Metaphor

Simple design

Refactoring

Short releases

Testing

Pair programming

Coding standards

Continuous integration

Collective ownership