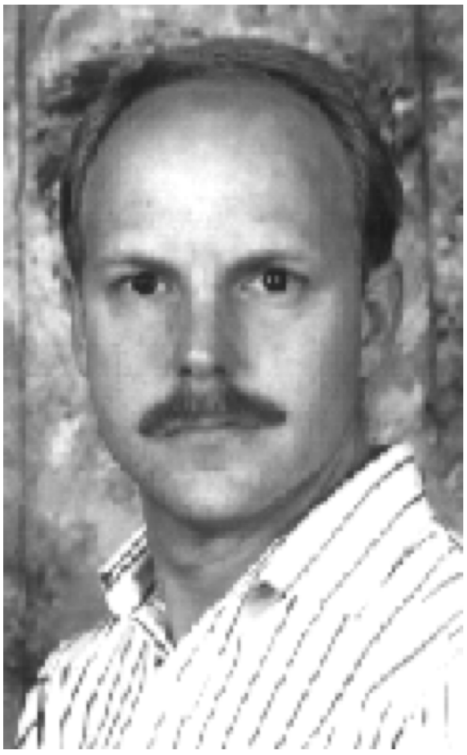
Extreme Programming

Shihong Huang

Department of Computer Science & Engineering
Florida Atlantic University

Kent Beck



- One of the pioneers of XP
- Worked together with Ron Jeffries
- Earlier work on Design Patterns

{Extreme}

An adjective and noun [C]: the largest possible amount or degree of something

E.g.: *extreme pain/stupidity/wealth*

-- definition from *Cambridge Advanced Learner's Dictionary*

What is XP? – 1

- It is one of several new *lightweight software methodology* for small to medium size team
- A software methodology is the set of rules and practice used to create computer programs
- A *heavyweight* methodology has many rules, practices, and documents
- A *lightweight* methodology has only a few rules and practices which are easy to follow
- It is a deliberate and disciplined approach to software development

What is XP? – 2

- It works by bring the whole team together in the presence of simple practices, two promises:
  - Programmers: work on things that matter; don't have to face scary situation alone; do their best to make system successful
  - Customers and managers: get most possible value out of every week; concrete progress on goals every few weeks; change direction w/o exorbitant costs
- "Extreme" takes commonsense principles and practice to extreme levels
  - Code reviews (pair programming)
  - Testing (unit testing, functional testing)
  - Design (refactoring)
  - Simplicity (the simplest thing that could possibly work)
  - Architecture (metaphor)
  - integration testing (continuous integration)
  - Short iterations (planning game)

XP Changes in the Way We Program

- Emphasize on testing, and test well
  - Another important issue to customers are bugs. XP emphasizes not just testing, but testing well. Tests are automated and provide a safety net for programmers and customers alike. Tests are created before the code is written, while the code is written, and after the code is written. As bugs are found new tests are added. A safety net of tight mesh is created. Bugs don't get through twice, and this is something the customers will notice
- Embrace changes of requirements
  - Another thing your customers will notice is the attitude XP programmers have towards changing requirements. XP enables us to embrace change. Too often a customer will see a real opportunity for making a system useful after it has been delivered. XP short cuts this by getting customer feed back early while there is still time to change functionality or improve user acceptance. Your customers are definitely going to notice this.

http://www.extremeprogramming.org/change.html

XP's Application

- Risky projects with dynamical change of requirements are perfect for XP
- In many software environments dynamically changing requirements is the only constant

XP Four Values

- XP improves software project in four essential ways:
  - Improve *Communication*
    - Managers, programmers, customers
  - Seek *Simplicity*: simple and clean design
  - Get *Feedback*: testing from day one
  - Proceed with *Courage*: deliver system earlier, response to change of requirement and technology

Communication

- Some problems with projects are due to the lack of communication
  - Programmer – Programmer (critical change)
  - Programmer – customer (ask the right question)
  - Manager – programmer
- Many circumstances can lead to a break down in communications
- XP keeps the right communication flowing by using: unit testing, pair programming, and task estimation

Simplicity

- "What is the simplest thing that could possibly work?"
- Simplicity is not easy
- It is hard not to look forward
- XP philosophy: to do a simple thing today and pay a little more tomorrow
- XP is making a bet!
- Simplicity and communication mutually support

Feedback

- Concrete feedback about system is priceless
- Optimism is an occupational hazard of programming – feedback is treatment
- Feedback works at different time scales
  - Minute and days: unit test for logic
  - Weeks and months: functional testing for all the descriptions implemented
- Works together with communication and simplicity

Courage

- When you discover there are no way forward, then fix the flaw – even it means to break half of your test
- Throw code away:
  - something isn't going well
  - your code is a little out of control in the end of day
  - ☞ Start over on most promising design

Major Practices in XP

- The planning game
- Small release
- Metaphor
- Simple design
- Testing
- Refactoring
- Pairing programming
- Collective ownership
- Continuous integration
- 40-hour week
- On-site customer
- Coding standards

Planning Game

- Quickly determine the scope of next release by combing business priorities and technical estimates
- Predict what will be accomplished by due date
- Determine what to do next
- Emphasis on steering the project

Small Release

- Put a simple system into production quickly
  → Release running, tested software, delivering business value chosen by customer, every iteration
- Release new version on a very short cycle
  → Web project release as often as daily
  → In house projects monthly or more frequently
- Every release should be as small as possible, containing the most valuable business requirements

Metaphor

- Guide all development with a simple shared story of how the whole system works
- Metaphor helps everyone on the project understand the basic elements and their relationships
- It replaces much of what other people called “architecture”
- An architecture is big boxes and connections

Simple Design

- The right design for the software at any given time is the one that:
  - Runs all the tests
  - Has no duplicated logic
  - States every intention important to the programmers
  - Has the fewest possible classes and methods
- Start simple and keep it that way
- Design exactly suited for the current functionality of the system
- No wasted motion, always ready for what is next
- Put in what you need when you need it

☞ Opposite from “Implement for today, design for tomorrow”

Testing

- Strong emphasis: XP puts testing at the foundation of development
- Not just testing, but test well
- Both Programmer and customer continually write tests
- Tests before, during and after code is written

Test-Driven Development

- Top XP teams practice “test-driven development”
- Working in very short cycles – adding a test, then making it work
- Teams produce code with nearly 100% test coverage, all the time – immediate feedback on how they are doing
- Before release any code to the repository, every single unit tests must run correctly

Refactoring

- Programmer restructure the system without changing its behavior
- To remove duplication, improve communication, simplicity, or add flexibility
- This means you do more work than necessary to get a feature running
- The purpose is to ensure add the next feature with reasonable amount effort
- Refactor when system asks to do so, not on speculation

Pair Programming

- All production code in XP is built by two programmers, sitting side by side
  - One partner coding
  - One partner thinking strategically
- Pairing dynamically
- Ensures code is reviewed by at least one other programmer → better design, testing, and better code
- Two heads really are better than one!

Collective Ownership

- Everybody takes responsibility for the whole of the system
- Not everyone knows every part equally well, although everyone knows something about every part.
- Any pair of programmers can improve any code anywhere at any time

→ All code gets the benefit of many people's attention

→ Contrast to: no ownership or individual ownership

Continuous Integration

- XP teams keep the system fully integrated at all times
- Every time a task is completed
- One XP team of 40 people builds at least eight or ten times a day!
- Have a machine dedicated to integration
- Integrating one set of changes a time – it is obvious who should fix a text that fails

40-Hour Week

- Overtime is a symptom of serious problem of project
- Keep fresh, creative, careful, confident
- XP rule: you can't work a second week of overtime
- If you put in too much extra hours, then you already have a problem that can't be solved by working more hours

On-Site Customer

- Include a real, live user on the team, available full-time to answer questions
- Objection: real users are too valuable to give. Manager have to choose:
  - Having software work sooner and better
  - Having the output of one or two people
- On-site customers still can do their normal work
- Downside: if project fails, they lose the work they did, plus the work they could have done if they hadn't been contributing to a failing project
- Cure: "only for a little while"

Coding Standards

- XP follows a common coding standards
- All code in the system looks as if it was written by a single – very competent – individual
- Specification of the standard are not important
- Important: all code looks familiar
  → In support of collective ownership

Resources

- www.extremeProgramming.org (Don Well)
- www.xprogramming.com (Ron Jeffries)
- "Extreme Programming Explained: Embrace Change" (Kent Beck, Addision-Wesley, 2000)