

Embracing Change with Extreme Programming

By Kent Beck

1

Outline

- ◆ The Need
- ◆ The Rules
- ◆ The Game
- ◆ Handling Problems
- ◆ Victories
- ◆ Opponents
- ◆ Take Caution
- ◆ Conclusions
- ◆ Discussion

2

The Need

3

Waterfall Model

- ◆ Depended upon clarity and correctness from users.
- ◆ Users didn't know what they wanted.
- ◆ Users requested "bells and whistles."

4

Waterfall Model

- ◆ The steps are not followed:

Activity	Phase			
	Design	Coding	Integration Testing	Acceptance Testing
Integration Testing	4.7	43.4	26.1	25.8
Coding	6.9	70.3	15.9	6.9
Design	49.2	34.1	10.3	6.4

5

Incremental Development

- ◆ "Grow" the system in a number of small steps.
- ◆ The user helps plan subsequent steps.
- ◆ Forces users to pick what is needed and eliminate "bells and whistles."

6

Incremental Development

- ◆ Since it proceeds in small steps it is easier to incorporate change.
 - Change in requirements
 - Change in functionality
 - Change in deadline
- ◆ It is ready for surprises.

7

Next Step: XP

- ◆ Extreme Programming takes the idea of incremental development to the next level.
- ◆ Plan, analyze, and design a little at a time.
- ◆ XP has 12 (or 13) simple rules.

8

The Rules

9

The Planning Game

- ◆ Schedule small tasks to be completed during the current iteration.
- ◆ Programmers will focus their attention on the tasks at hand.
- ◆ List of tasks is updated regularly.

10

Small Releases

- ◆ A functional system is produced after a few months.
- ◆ System is released before the whole problem is solved.
- ◆ New releases regularly (daily to monthly).

11

Metaphor

- ◆ Use metaphors to describe how the system should work.
- ◆ These analogies express the functionality of the system.

12

Simple Design

- ◆ The code should pass all tests and fulfill certain functionality while maintaining:
 - No duplicate code.
 - Fewest possible classes and methods.
 - "Say everything once and only once."

13

Tests

- ◆ Tests are continuously written with the system.
- ◆ All tests are run together at every step.
- ◆ Customers write tests that will convince them the system works.
- ◆ Don't proceed until current system passes ALL tests.

14

Refactoring

- ◆ The code may be changed at any time to provide:
 - Simplification
 - Flexibility
 - Reduced redundancy

15

Pair Programming

- ◆ All programming is done with two coders at the same machine.
- ◆ The programmers must share one mouse, keyboard, screen, etc.

16

Continuous Integration

- ◆ Newly finished code is integrated immediately.
- ◆ System is rebuilt from scratch for every addition.
- ◆ New system must pass all tests or new code is discarded.

17

Collective Ownership

- ◆ All workers can access any of the code.
- ◆ Any programmer can change any part of the system if an opportunity for improvement exists.

18

On Site Customer

- ◆ At least one customer is always present.
- ◆ This customer is available full-time to answer questions about the system.

19

40 Hour Weeks

- ◆ Consecutive weeks of overtime is not allowed.
- ◆ The need for overtime is a symptom of a deeper problem.

20

Open Workspace

- ◆ Work on computers set up in the middle of a large room with cubicles around the edges.

21

Coding Standards

- ◆ Agree upon standards for coding styles.
- ◆ Promotes ease of understanding and uniformity.

22

Just Rules

- ◆ These rules are just rules.
- ◆ XP teammates agree to follow all of the rules.
- ◆ An agreement can be made to change the rules.
 - Must address side effects of rule change.

23

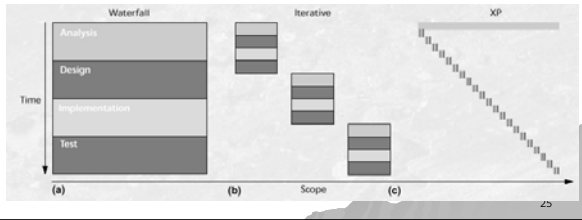
The Game

XP: How it Works

24

Development Cycle

- ◆ Developers estimate development time and cost of each feature.
- ◆ Customer picks most important features to be implemented first.



Development Cycle

- ◆ Coders pick the features (stories) on which they want to work.
- ◆ Coders write test cases to show their code works.
- ◆ System is evolved at every step to maintain simplicity.

Stories

- ◆ Stories are what XP calls features.
- ◆ Stories are listed during initial planning.
- ◆ A story is described on an index card.

Stories

- ◆ Stories must be:
 - Business oriented
 - Testable
 - Estimable
- ◆ Should be feasibly finished in a few days.

Release

- ◆ Customer chooses most important stories that make sense together.
- ◆ Prevents users from asking for "bells and whistles" that aren't as important.
- ◆ These will be implemented first.

Release

- ◆ Like going to grocery store with \$100.
 - Pick what you want until it adds up.
- ◆ There are two options:
 - Programmers calculate finish date.
 - User picks a date.

Iteration

- ◆ The goal of each iteration is to add features to the system.
- ◆ Customer chooses most important stories of what remains.
- ◆ Technical tasks are added to the list of stories.

31

Iteration

- ◆ Programmers choose their task and estimate a time frame.
 - If one coder has a smaller time frame, he chooses more tasks.
- ◆ Tests and Tasks are added to the system.
- ◆ At the end of each iteration, the system passes all of the tests.

32

Tasks

- ◆ To implement a task, the coder meets with customer.
- ◆ The partners create a list of tests their code must pass.
- ◆ Then they code and see if it passes their tests.
- ◆ Refactoring of the system is done when necessary.

33

Test

- ◆ Testing is fundamental to XP.
- ◆ Programmers write tests for their own code.
- ◆ Tests are written before the code is.
- ◆ Teaches coders what their code should do.

34

Test

- ◆ Tests are not thrown away when the system passes them.
- ◆ Tests are kept and reused.
- ◆ These tests will be run on the entire system after each integration.
- ◆ The customer tests are added as well.
- ◆ Provides confidence in the code.

35

Handling Problems

36

Underestimation

- ◆ Sometimes too great a commitment will be made.
- ◆ Check to see if rules are being followed.
- ◆ If stories cannot be completed, ask the user to choose a subset.
 - Other stories will be finished later.

37

Uncooperative Customers

- ◆ Some customers won't play the game.
- ◆ XP relies on trust.
- ◆ Don't move on based on guesses.
- ◆ If customer never makes an effort, perhaps the system isn't worth being built.

38

Turnover

- ◆ If a programmer leaves, they don't take any information that only they have.
- ◆ Tests exist for every feature, so nothing can be broken by ignorance.
- ◆ New people can be trained by pairing with experienced programmers.

39

Changing Requirements

- ◆ This isn't a problem for XP as it is for other development models.
- ◆ Have only planned for today, won't have to change our plans.
- ◆ New features will just be added to the stories.

40

Success Stories

41

Chrysler's Payroll System

- ◆ 10 programmers, 4 years.
- ◆ First use of XP.
- ◆ Seen as a monument of success for XP.
- ◆ They used XP to rework the internals of a large payroll system.

42

Ford Motor's Cost Analysis System

- ◆ 12 programmers, 6 years.
- ◆ Had constantly changing requirements.
- ◆ Small releases were beneficial.
- ◆ Customers and Managers noticed greater system stability.

43

Critics

- ◆ Steve McConnell said that different processes are needed for different projects.
 - Also said that few of the ideas are new.
- ◆ Doug Rosenberg points out that the Chrysler System was actually a failure.
 - It was cancelled before completion.
 - It proved capable of less than a quarter of what was expected of it.

44

Warnings

- ◆ Beck admits that it would take courage, flexibility, and a "willingness to abandon the project" to use XP.
- ◆ Beck recommends applying it to small to medium sized projects rather than large ones.
- ◆ Also recommends using it on sub-problems first to get the hang of it.

45

Conclusions

- ◆ XP appears to be a good approach to building a system that may have changing requirements.
- ◆ May not apply to a system where planning ahead is necessary.

46

Discussion

- ◆ Does it sound like it would work?
 - Why?
- ◆ Which of its rules make sense?
- ◆ Is pair programming a good idea?

47

Bibliography

- ◆ K. Beck, Embracing Change with Extreme Programming. IEEE Computer. 1999.
- ◆ M. Baer, The New X-Men. Wired Magazine (p124-9). Sept 2003.
- ◆ H. Van Vliet, Software Engineering Principles and Practices. John Wiley & Sons. New York, NY. 2002.

48