

Extreme Programming

Introduction

What's wrong with software today?

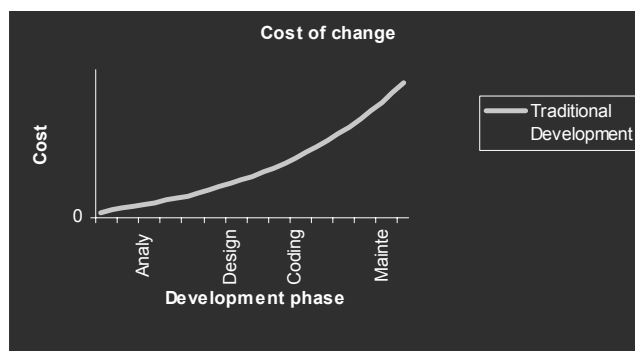
- Software development is risky and difficult to manage
- Customers are often dissatisfied with the development process
- Programmers are also dissatisfied

Extreme Programming Defined

- Extreme Programming (XP) is a discipline of software development based on values of simplicity, communication, feedback, and courage.
- It works by bringing the whole team together in the presence of simple practices, with enough feedback to enable the team to see where they are and to tune the practices to their unique situation.
- Why does XP help?

Current Processes are Heavy

- Traditional methodologies seek to reduce costs according to Boehm's cost of change curve



Boehm's Curve

- To accomplish this:
 - We need lots of up front planning, resulting in “heavy” methodologies
 - Every bug caught early saves money, since models are easier to modify than code
 - Large investments are made in up front analysis and design models, because of the cost of late error discovery
 - This leads to a waterfall mentality with BDUF (Big Design Up Front)
- But that logic is based on development in the 1970's and 1980's

Things have changed a little...

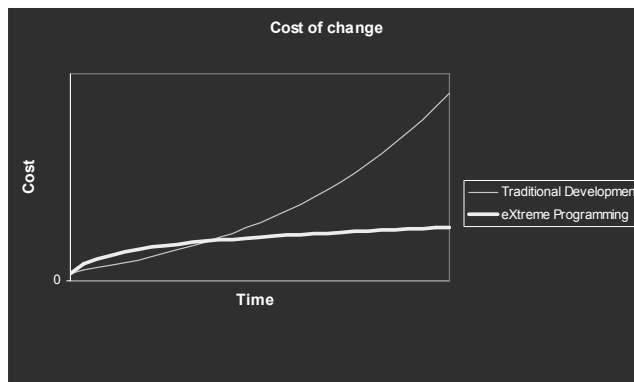
- Computing power has increased astronomically
- New tools have dramatically reduced the compile/test cycle
- Used properly, OO languages make software much easier to change
- The cost curve is significantly flattened, i.e. costs don't increase dramatically with time
- Up front modeling becomes a liability – some speculative work will certainly be wrong, especially in a business environment

Why XP Helps

- Extreme Programming is a “light” process that creates and then exploits a flattened cost curve
- XP is people-oriented rather than process-oriented, explicitly trying to work with human nature rather than against it
- XP Practices flatten the cost of change curve.

Why XP Helps continued

- The XP Cost of Change Curve



Why XP Helps continued

- “Software development is too hard to spend time on things that don't matter. So, what really matters? Listening, Testing, Coding, and Designing.”

(Kent Beck, “father” of Extreme Programming)

- XP focuses on what “really matters” in software development.

Economics of XP

- Promotes incremental development with minimal up-front design
- Results in a “pay as you go” process, rather than a high up-front investment
- Delivers highest business value first
- Provides the option to cut and run through frequent releases that are thoroughly tested

Economics of XP continued

- XP tends to use small teams, thus reducing communication costs.
- XP puts Customers and Programmers in one place.
- XP prefers index cards to expensive round-trip UML diagramming environments
- XP's practices work together in synergy, to get a team moving as quickly as possible to deliver value the customer wants

Customer Bill of Rights

- As the customer, you have the right to:
 - An overall plan, to know what can be accomplished, when, and at what cost;
 - Get the most possible value out of every programming week;
 - See progress in a running system, proven to work by passing repeatable tests that you specify;
 - Change your mind, to substitute functionality, and to change priorities without paying exorbitant costs;
 - Be informed of schedule changes, in time to choose how to reduce scope to restore the original date, even cancel at any time and be left with a useful working system reflecting investment to date.

Developer Bill of Rights

- As the Developer, you have the right to:
 - Know what is needed, with clear declarations of priority;
 - Produce quality work at all times;
 - Ask for and receive help from peers, superiors, and customers;
 - Make and update your own estimates;
 - Accept your responsibilities instead of having them assigned to you.

XP Values

- XP is based on 4 core values:
 - Communication
 - Simplicity
 - Feedback
 - Courage

XP Roles

- **Customer**
 - Writes User Stories and specifies Functional Tests
 - Sets priorities, explains stories
 - May or may not be an end-user
 - Has authority to decide questions about the stories
- **Programmer**
 - Estimates stories
 - Defines Tasks from stories, and estimates
 - Implements Stories and Unit Tests
- **Coach**
 - Watches everything, sends obscure signals, makes sure the project stays on course
 - Helps with anything
 - Applies “Rolled Up Newspaper” as required

XP Roles continued

- **Tracker**
 - Monitors Programmers' progress, takes action if things seem to be going off track.
 - Actions include setting up a meeting with Customer, asking Coach or another Programmer to help
- **Tester**
 - Implements and runs Functional Tests (not Unit Tests!)
 - Graphs results, and makes sure people know when test results decline.
- **Doomsayer**
 - Ensures that everybody knows the risks involved
 - Ensures that bad news isn't hidden, glossed over, or blown out of proportion

XP Roles continued

■ Manager

- Schedules meetings (e.g. Iteration Plan, Release Plan), makes sure the meeting process is followed, records results of meeting for future reporting, and passes to the Tracker
- Possibly responsible to the Gold Owner.
- Goes to meetings, brings back useful information
- Pays for pizza

■ Gold Owner

- The person funding the project, which may or may not be the same as the Customer

XP Practices

- XP is based on 12 key practices:
 - The Planning Process
 - Release Planning
 - Iteration Planning
 - Frequent, Small Releases
 - System Metaphor
 - Simple Design
 - Test Driven Development
 - Refactoring

XP Practices continued

- XP is based on 12 key practices (continued):
 - Pair Programming
 - Collective Code Ownership
 - Continuous Integration
 - Sustainable Pace
 - On-site Customer
 - Coding Standard

Why 'Extreme'?

- Reflects the intensity and fearlessness of athletes in extreme sports
- The XP practices get "turned up" to a much higher "volume" than on traditional projects.

Stages of an XP project

- Initiation
 - User Stories
- Release Planning
- Release (each Release is typically 1 - 6 months)
 - Iteration 1 (typically 1 - 3 weeks)
 - Development
 - Deployment
 - Acceptance Testing
 - Iteration 2
 - Development
 - Deployment
 - Acceptance Testing
 - :
 - Iteration n

Gathering Requirements

- Responsibilities
 - Key Point: The Customer is responsible for the requirements.
 - Programmers help to gather and clarify requirements. Customers especially need help with non-functional requirements and with working out the details of acceptance tests.
- Documentation
 - User Stories
 - Acceptance Test Cases

User Stories

- A short description of the behavior of the system from the point of view of the Customer
- Use the Customer's terminology without technical jargon
- One for each major feature in the system
- Must be written by the users
- Are used to create time estimates for release planning
- Replace a large Requirements Document

User Stories continued

- Drive the creation of the acceptance tests:
 - Must be one or more tests to verify that a story has been properly implemented
- Different than Requirements:
 - Should only provide enough detail to make a reasonably low risk estimate of how long the story will take to implement.
- Different than Use Cases:
 - Written by the Customer, not the Programmers, using the Customer's terminology
 - More "friendly" than formal Use Cases

User Stories continued

User stories have three crucial aspects:

- Card
 - Enough information to identify the story
- Conversation
 - Customer and Programmers discuss the story to elaborate on the details
 - Verbal when possible, but documented when required
- Confirmation
 - Acceptance tests to confirm that the story has been properly implemented

User Story Examples

A user wants access to the system, so they find a system administrator, who enters in the user's First Name, Last Name, Middle Initial, E-Mail Address, Username (unique), and Phone Number.

Risk: Low

Cost: 2 points

User Story Examples continued

The user must be able to search for a book.

Risk: High

Cost: (too large!)

User Story Examples continued

The user must be able to search for a book by Title, and display the results as a list.

Risk: Med.

Cost: 1 point

User Story Examples continued

The user must be able to search for a book by Author, and display the results as a list.

Risk: Med.

Cost: 1 point

User Story Examples continued

The user must be able to search for a book by ISBN number, and display the results as a list.

Risk: Med.

Cost: 1 point

User Story Examples continued

The user must be able to search for a book by *Category*, and display the results as a list.

Risk: Med.

Cost: 2 points

Acceptance Tests

- Formal test to determine if a system satisfies its acceptance criteria, i.e. the User Stories!
- Should be automated, but may simply be a series of repeatable steps
- At least one Acceptance Test for each Story

Release Planning

- Customer defines the business value of desired features (User Stories)
- Programmers provide estimates of 1, 2 or 3 “points”
- Stories larger than 3 points must be split into smaller stories
- Customer decides which Stories are to be included in a Release
- Focus on completing the Stories with the highest business value and highest risk first

Release Planning continued

- Stories for a Release are arranged into 1-3 week Iterations
- Higher risk, and higher priority stories in earlier Iterations
- For new system, the 0th Iteration defines the basic skeleton of the application and infrastructure required
- The Release and Iterations have fixed dates for completion – dates are fixed, scope is variable
- This is the Release Plan

Iteration Planning

- Stories for Iteration are broken down into Tasks by Programmers
- Tasks are estimated by all Programmers as a group
- Programmers “sign up” for Tasks, and estimate the time to complete
- Can only sign up for as many points as were completed in the last Iteration
- Once development begins, Project Velocity measures progress

Programmer Tests

- Automated tests written to test the behaviour of individual classes
- Fundamental to XP, and maintaining a flat cost curve
- XP uses a Test First mentality; write the test, then write to code to make the test pass.
 - "Never write a line of code without a failing test." (Kent Beck)
- No code goes into production unless it has associated tests
- Tests are written first
- Tests determine what code you need to write

Programmer Tests continued

- Programmer Tests must run at 100% before code is integrated
- At most one test failing at any time
- “Grey-box” testing
- Assist with Refactoring (promote Courage)
- Testing frameworks exist for many languages:
 - JUnit for Java
 - CPPUNIT for C++
 - NUnit for all .Net languages

Sample JUnit Test

- First, write the test:

```
import junit.framework.TestCase;

public class TestMovieList extends TestCase {
    public void testEmptyList() {

        MovieList emptyList = new MovieList();

        assertEquals( "Empty list should have size of 0",
            0, emptyList.size() );
    }
}
```

Sample JUnit Test continued

Now write the code to make the test pass.

- First, there is no MovieList class:

```
public class MovieList {  
}
```

- Now we need a 'size' method that returns an 'int':

```
public class MovieList {  
    public int size() {  
        return 0;  
    }  
}
```

Refactoring

- “Refactoring is the process of changing a software system in such a way that it does not alter the external behavior of the code yet improves its internal structure.” (Martin Fowler)
- Improves the quality and maintainability of code
- Supported by the Programmer Tests -
Programmers can quickly verify that refactoring has not created regression errors

Sample Refactoring

■ Extract Method

- You have a code fragment that can be grouped together. Turn the fragment into a method whose name explains the purpose of the method.

```
void printOwing() {
    printBanner();
    // Print Details
    System.out.println( "Name " + _name);
    System.out.println( "Amount " +
        getOutstanding());
}
```

Sample Refactoring continued

■ The result:

```
void printOwing() {
    printBanner();
    printDetails( getOutstanding() );
}

void printDetails (double outstanding) {
    System.out.println( "Name: " + _name);
    System.out.println( "Amount: " + outstanding);
}
```

(Source: "Refactoring", by Martin Fowler)

Transitioning to XP

- “We can’t use all of the Practices!” or “We can’t throw everything out and go XP!”
 - Onsite Customer
 - Planning Game
 - Programmer Tests and Test Driven Development
 - Coding Standard and Continuous Integration
 - Simple Design and Refactoring
 - Frequent, small Releases
 - Pair Programming
 - Collective Code Ownership
 - Sustainable Pace
 - System Metaphor

Review

- Open, honest communication
- Rapid feedback at all levels
- Quality Work
- Assume Simplicity
- Incremental Change
- Small initial investment
- Embrace Change
- Travel light
- Local adaptation

References and Resources

- <http://www.xprogramming.com> - Ron Jeffries' site for all things Extreme!
- <http://c2.com/cgi/wiki?ExtremeProgramming> - Extreme Programming Wiki; the ultimate source of Extreme Programming information!
- <http://www.refactoring.com> - The primary site for Refactoring.
- <http://www.martinfowler.com> - Martin Fowler's site, with many writings and examples about Refactoring, XP and Design Patterns.
- <http://www.saorsa.com/page.php?page=pgtdd> - A Practical Guide to Test-Driven Development (Saorsa Development)
- <http://www.agilealliance.org/> - The AgileAlliance is dedicated to promoting the concepts of agile software development.
- <http://crystalmethodologies.org/articles/panlc/peopleasnonlinearcomponents.html> - Alistair Cockburn's research paper on the effect of People on software development
- <http://cseng.aw.com/catalog/series/0,3841,13,00.html> - Addison-Wesley XP Series Books; Extreme Programming books of all colours!