Lecture 4: Extreme Programming

CS190: Software System Design

January 30, 2002
Steven P. Reiss

I. Today's Class

A. Overview of XP as it involves software engineering

1. Contrast to classical techniques discussed last time

2. Provide a sense of motivation and expectations

B. Any time left over will be for group meetings

1. How much time is needed for group meetings

C. Next Monday

1. I'd like each group to do a 15 minute presentation of their proposed project

a) Describe what it is, why it is, who the customer is

b) Provide a few stories of what it will do.

II. What's wrong with classical SE

A. It has the wrong orientation

1. Currently it is geared to the programmer

a) Hard to make it adapt to the customer -- customer input at the beginning is incomplete and too early

b) Doesn't focus on change. Software engineering should be about change. That's what maintenance is all about; that is what the rest of development should be aimed at.

c) Designed to get things right the first time

(1) Is this possible

(2) Is this desirable

2. Should have a different focus

a) The software must meet the customer's needs, even if they change

b) The software must work and do what it is supposed to

c) The software must be maintainable

B. It is unrealistic

1. User requirements are a moving target

a) User's rarely know what they want

b) User's wants change over time

c) User's wants change as they understand what they can get.

2. Design can't anticipate the needed changes

a) Blind men with the elephant

b) Any design done without knowing the full requirements will have to be changed

c) A better program will result from having the right design at the end than having the wrong design from the beginning

3. The system must work completely

a) Testing should not wait until the end; it should be done throughout to ensure bugs are found as soon as possible and when the code is fresh in ones mind

b) Integration should not wait until the end; it should be done throughout to avoid all the problems and delays inherent to it

4. Programmer's can't work in isolation

a) Need feedback from themselves and users

b) User feedback throughout development yields a more useful system

c) Need to release new versions of the system frequently to get such feedback

5. Can't depend on individuals

a) People get sick, quit, get fired, ...

b) Individuals shouldn't own pieces of the software; everyone should own everything

c) Everyone can change everything; anyone can make bug fixes and make things work

III. Whither Extreme Programming

A. Suppose we want to address these issues

1. **We should have a heavy emphasis on the customer**
2. **We should emphasize code rather than design**
3. **We should emphasize features rather than classes**
4. **We should emphasize testing**
5. **We should emphasize having a working system**
6. **We should emphasize group responsibility**

B. XP is an attempt to do this

1. **Emphasis on the customer/user throughout**
   - a) Keep the customer involved via stories and conversation
   - b) Keep the customer involved with frequent releases and feedback
   - c) Keep the customer involved in determining priorities for what comes next, what works and doesn’t, etc.
2. **Emphasis on code rather than design**
   - a) Get something up and running quickly
   - b) Keep the code as simple as practical
   - c) Continually modify the code so it matches the best design
     - (1) Take the design from experience with the code
     - (2) But don’t freeze either the code or the design
   - d) Try out different alternatives as you go
3. **Emphasis on features**
   - a) Stories provide an understandable interface to the customer and user
   - b) Stories represent particular features in the application
   - c) Features provide a reasonable basis for building the system up piece by piece
   - d) Provide a better incremental foundation than fully-functional classes
   - e) Provide an easier means for modifying or adapting functionality
4. **We should emphasize testing**

a) This is one of the few ways of ensuring some sort of correctness of code that is frequently changing

b) This gives other programmers confidence in all the code, even if they didn't write it

c) This lets programmers modify significant fractions of the code and have a sense of when the modifications work

5. We should emphasize having a working system

a) Integration is very difficult; continuous integration makes the problem seem a lot easier

b) Continuous integration forces people to learn all aspects of the system and thus have ownership stake

c) Working system with frequent releases provides positive feedback to developers

d) Working system provides positive feedback to users

e) Working system provides a basis for customer feedback

6. We should emphasize group responsibility

a) If everyone owns the code, then loss of one person will not cripple the project

b) If everyone owns the code, bug fixes and integration will go a lot faster

c) Pair programming, continual integration, cross-cutting features all help to build group responsibility

IV. XP as a process model

A. Recall the phases of software engineering

1. Show how they fit together in XP

V. XP is CS190

A. Requirements -- overview of projects

1. Next Monday; with 15 minute presentations

B. Specifications -- stories and an initial set of features

1. Monday a week; with 45 minute presentations

C. **Top Level Design -- more stories plus a model of how the overall system will be approached by the team**
   1. **Wednesday a week later**

D. **Coding**
   1. **Done with and after top-level design**
   2. **Emphasis on testing and continual integration**
   3. **Use of pair programming**

E. **Initial System Release**
   1. **Friday 3/1; first set of features implemented**
   2. **In-class demos**
   3. **Customer and class feedback**

F. **Incremental System Releases**
   1. **Every Friday thereafter, with demos**
   2. **Customer and class feedback**
   3. **Discussion of priorities of what comes next**