Extreme Programming

Sergey Konovalov and Stefan Misslinger

May 23, 2006

Contents

1 Introduction

Extreme Programming (XP) is a system of practices that a community of software developers is evolving to address the problems of quickly delivering quality software, and then evolving it to meet changing business needs. It has already been proven at many companies of all different sizes and industries world wide.

Why it is called "Extreme"?

- Taking proven practices to the extreme
- If testing is good, let everybody test all the time
- If code reviews are good, review all the time
- If design is good, refactor all the time
- If integration testing is good, integrate all the time
- If simplicity is good, do the simplest thing that could possibly work
- If short iterations are good, make them really, really short

2 Why do we need XP?

Classic approaches to software development have a set of well-know problems:

- Schedule slips
- Business misunderstood
- Defect rate
- Management
- Motivation of developers

These problems become critical in some fields of software development. For example, a business project that is going to capture a new market niche by using a new technology. The challenge in such scenarios can be really hard and even a small delay with the product release can be fatal.

Extreme Programming is intended to solve these and some other problems of "classic" software development. One should understand however that the problems are tied very closely thus complex solutions are required.

Schedule slips: These are the most common problem of software projects. Usually the roots of the problems are changes in requirements and an underestimation of the project scope. XP solution: One of the XP principles are short iterations. It helps to keep the schedules accurate and keep the project ready for changes in requirements.

Business misunderstood: Very often business (customer) is not able to explain what is actually needed using the technical language. A common practice is that the business part of the project communicates with developers only in the initial and final phase of the project life-cycle. Sometimes it leads to the serious problem, that the business gets a product that was not actually needed. XP solution: Let the business be a part of the team. Customers should always be available for developers and vise versa. Small releases allow business to understand the project direction and to correct it if necessary.

Defect Rate: A lot of defects can kill any software project. Usually, the cause of a high defect rate is forcing the developers to work faster (e.g. when the team is behind schedule). Thus quality is sacrificed for speed. It is the worst thing that managers could make, because bugfixing will require much more time. XP solution: Testdriven development and Continuous testing. Testing must be an essential part of the project.

Management: Sometimes management costs overlap the cost of development. Effective and cheap management is one of the most important things in software projects. XP solution: XP

suggests using shared understanding and people rotation. Thus people will have a shared view of the project in which they work. It also allows to bypass the bottle neck problem.

Motivation of developers: It is really difficult and not an unequivocal task. XP solution: XP is trying to reconcile humanity and productivity. People like XP because it encourages communication, cross training, people rotation and etcÉ

3 Economics of Software Development

A critical concept of software development is the cost of change. Figure 1 shows the traditional cost of change curve for the single release of a project following a serial (waterfall) process. It shows the relative cost of addressing a changed requirement, either because it was missed or misunderstood, throughout the lifecycle. It is obvious that the cost of fixing errors increases exponentially the later they are detected in the development life cycle, because the errors within a serial process build on each other.

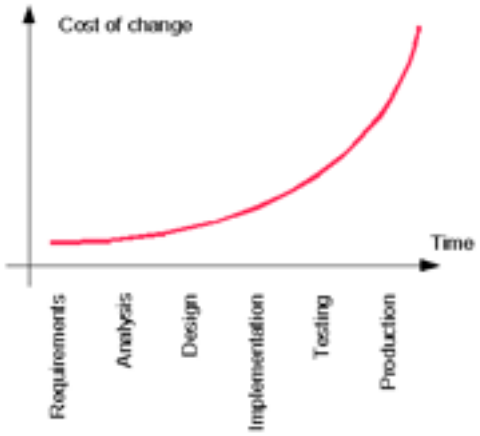

Figure 1:

As you can see in Figure 1 it is necessary to integrate testing into each moment of the development life cycle. By reducing the feedback loop, the time between creating something and validating it, you will clearly reduce the cost of change - Figure 2. In fact, some authors argue that in Extreme Programming the cost of change curve is flat.

The main point is that the feedback loop is dramatically reduced in XP. It is obtained by the use of a test-driven development (TDD) approach. Here the feedback loop is effectively reduced to minutes instead of days, weeks, or even months which is typical in serial processes. As a result the chances are good, that the cost of change will not get out of hand.

4 Extreme Programming Values

The following are the basic values of Extreme Programming:

- Communication
- Simplicity
- Feedback
- Courage

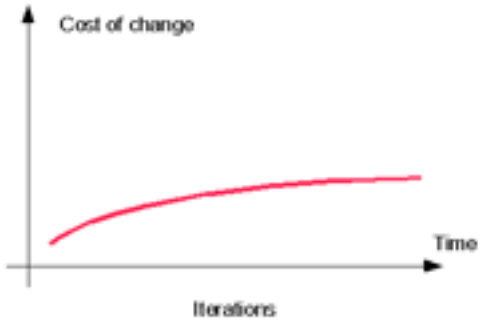

Figure 2:

These values are tied to each other and can be seen as a tetrahedron - Figure 3.

Figure 3:

Communication is an essential part of any development process. In formal software development methodologies, this task is accomplished through documentation. Extreme Programming techniques favor rapid disseminating project information among members of a development team. The goal is to give all developers a shared view of the system which matches the view held by the users of the system. Extreme Programming favors a metaphor, collaboration of users and programmers, frequent verbal communication and feedback. One of the main XP goals is to create an integrated team consisting of development and customers (business).

Feedback is a critical component of Extreme Programming. It is used in different aspects of the development:

- Feedback from System. It is a way of determining the current state of the system.
- Feedback from customers. It is a way to make sure that the project is going to the right direction.
- Feedback from developers. When customers come up with new requirements in the planning game the team directly gives an estimation of the time that it will take to implement.

A good explanation of *Simplicity* is given in Extreme Programming Explained book:
"XP is making a bet. It is betting that it is better to do a simple thing today and pay a little more tomorrow to change it if it needs it, than to do a more complicated thing today that may never be used anyway."
XP asks for each team member, "What is the simplest thing that could possibly work?" Make it simple today, and create an environment in which the cost of change tomorrow is low.

Courage enables developers to feel comfortable with refactoring their code when necessary. This means reviewing the existing system and modifying it so that future changes can be implemented more easily.

Continuous testing for example provides the developers with the necessary courage and confidence to make these changes. Running your unit tests after code is added or changed gives you a confirmation that you did not brake something.

5 Extreme Programming Rules and Practices

Extreme Programming has a set of well-proven rules and practices that cover each aspect of the software development.

5.1 XP Planning

XP Planning is also known as XP Planning Game. But there are no lost in the game. Business and development cooperate to produce the maximum business value as rapidly as possible. The planning game happens at various scales, but the basic rules are always the same:

1. Business comes up with a list of desired features for the system. Each feature is written out as a User Story, which gives the feature a name, and describes in broad strokes what is required.

2. Development estimates how much effort each story will take, and how much effort the team can produce in a given time interval (the iteration).

3. Business then decides which stories to implement in what order, as well as when and how often to produce small releases of the system.

5.2 XP Designing

The key point of XP Designing is simplicity. The rule is "always use the simplest possible design that gets the job done".

All members of the team should choose one *system metaphor* to keep the team on the same page by naming classes and methods consistently. What you name your objects is very important for understanding the overall design of the system and code reuse as well. Being able to guess at what something might be named if it already existed and being right is a real time saver. Choose a system of names for your objects that everyone can relate to without specific, hard to earn knowledge about the system.

Sometimes during the planning game or at other project stages there is a need to answer to a tough technical or design questions than cannot be answered immediately because the lack of information (e.g. when new technologies are involved). Extreme Programming suggests to use a *spike solution*. It is a simple program to explore potential solutions that addresses only one problem at a time. Most spikes are not good enough to keep and integrate it into the project, so the developer should expect to throw it away.

Test-driven development provides programmers with a unique opportunity to *refactor* whatever and whenever possible. Refactoring helps to keep the design simple as you go and to avoid needless clutter and complexity. Keeping the code clean and concise so it is easier to understand, modify, and extend. Developer should make sure that everything is expressed once and only once.

5.3 Coding

The following are the main coding rules and practices of Extreme Programming.

Customer on site. Perhaps the most extreme of the XP principles is its insistence on having a real customer working directly with the project. This person should be one of the eventual users of the system. All phases of an XP project require communication with the customer, preferably face to face, on site.

Coding standards. As in XP everyone is allowed to modify the communal code, coding standards are necessary. They keep the code consistent and easy for the entire team to read and refactor.

Pair Programming. All code to be included in a production release is created by two people working together at a single computer. Pair programming increases software quality without impacting time to deliver. It is counter intuitive, but 2 people working at a single computer will add as much functionality as two working separately except that it will be much higher in quality. With increased quality comes big savings later in the project. The best way to pair program is to just sit side by side in front of the monitor. Slide the key board and mouse back and forth. One person types and thinks tactically about the method being created, while the other thinks strategically about how that method fits into the class.

Simplicity. Always do the simplest thing that works. This may not always be the optimal solution, but the optimization can be left till last. Make it work, make it right, then make it fast.

Continuous Integration. Developers should be integrating and releasing code into the code repository every few hours, when ever possible. The unit tests have to run 100% both before and after integration. With this you can guarantee, that you always have a releasable codebase. Integration is a "pay me now or pay me more later" kind of activity. That is, if you integrate through out the project in small amounts you will not find your self trying to integrate the system for weeks at the project's end while the deadline slips by.

5.4 Testing

Testing is another integral part of XP. XP teams focus on validation of the software at all times.

Unit Tests. A unit test is a procedure used to validate that a particular module of source code is working properly. Unit tests are released into the code repository along with the code they test. Code without tests may not be released. If a unit test is discovered to be missing it must be created at that time. Unit tests enable collective code ownership. When you create unit tests you guard your functionality from being accidentally harmed. Requiring all code to pass all unit tests before it can be released ensures all functionality always works.

Code the Unit Test First. If you create your test before the code, you will find it much faster and easier to create your code. Creating a unit test helps a developer to really consider what needs to be done. Requirements are nailed down firmly by tests. There can be no misunderstanding a specification written in the form of executable code.

Acceptance tests. Acceptance tests are created from the user stories. Acceptance tests are black box system tests. Each acceptance test represents some expected result from the system. Acceptance tests should be automated so they can be run often.

6 References

- Don Wells, 2001. Extreme Programming: A gentle introduction. [online]. Available at: http://www.extremeprogramming.org/

- Ronald E Jeffries, 2006. XProgramming.com - an Agile Software Development Resource. [online].
  Available at: http://www.xprogramming.com/
- John Brewer, 2001. Extreme Programming FAQ. [online]. Jera Design.
  Available at: http://www.jera.com/techinfo/xpfaq.html
- Scott W. Ambler, 2006. Examining the Agile Cost of Change Curve [online].
  Available at: http://www.agilemodeling.com/essays/costOfChange.htm
- Wikipedia, 2006 Extreme Programming [online]. Wikipedia.
  Available at: http://en.wikipedia.org/wiki/Extreme_Programming
- D. Astels, G. Miller, M. Novak, 2002. A practial guide to eXtreme programming
- J. Highsmith, 2000. Extreme Programming, Agile Project Management Advisory Service White Paper Steve McConnell, Code Complete, Second Edition, Microsoft Press, Redmond, 2004.