International Journal of Information Technology and Knowledge Management
July-December 2010, Volume 3, No. 2, pp. 699-705

EXTREME PROGRAMMING: NEWLY ACCLAIMED AGILE SYSTEM DEVELOPMENT PROCESS

Er. Rohini Mahajan[1] & Er. Pawanpreet Kaur[2]

Extreme Programming is one of the most discussed subjects in the software development community. It is one of the several popular agile approaches to software development that stresses customer satisfaction and teamwork. Extreme Programming (or XP) XP delivers clean designs and high quality software on a realistic schedule. Extreme Programming (or XP) is a set of values, principles and practices for rapidly developing high-quality software that provides the highest value for the customer in the fastest way possible. But what makes XP extreme? And how does it fit into the new world of agile methodologies? This paper will first give introduction to Extreme Programming. Then it will give the main problems before extreme programming that were in Agile Software Development. The paper also explains how XP uses a set of practices to build an effective software development team that produces quality software in a predictable and repeatable manner. The Extreme Programming movement has been a subset of the object-oriented (OO) programming community for several years, but has recently attracted more attention. This paper is written with the intent to help software developers understand the key concepts of this methodology and to provide a framework for implementing these processes into a business.

1. Introduction

Extreme Programming is a relatively new, somewhat controversial development process that takes many known software development practices. Extreme Programming, or XP, is a lightweight discipline of software development based on principles of simplicity, communication, feedback, and courage. XP was developed to address the needs of small teams who are confronted with vague and changing requirements. XP is a discipline of software development that follows a specific structure that is designed to simplify and expedite the process of developing new software. XP takes many of its fundamentals from other iterative development methodologies, including RAD and JAD. However, unlike those methodologies, which are more of a rapid prototyping approach, XP creates individual components that can be quickly developed and integrated into a larger software system. “Extreme” means these practices get “turned up” to a much higher “volume” than on traditional projects.The result is stable, productive, and very rapid because the practices support each other the more they are used together without interference. An Extreme project is typically so stable and sedate it can lead to FortyHourWeeks without any schedule slips.

Extreme Programming codifies a set of practices that many software developers are willing to adopt in both action and spirit. Many of these practices are grounded in fundamental project management theory. When software development teams embrace the practices of Extreme Programming an opportunity is created for a broad set of project management practices to become meaningful and accessible to the developers, while at the same time making clear,. However, it is important to acknowledge that Extreme Programming is not a comprehensive project management system, but rather is a set of software development best practices that overlap nicely with best practices from the project management domain.

Lecturers, SUSCET, Tangori (Mohali)
er.rohinimahajan@gmail.com[1], er.pawanpreet@gmail.com[2]

2. Problems with Agile Software Development

Agile Software Development is a conceptual framework for software engineering that promotes development iterations through out the life-cycle of the project. Mostly techniques under agile software development minimize risk by developing software in short amounts of time. Each iteration lasts one week to four weeks. Each iteration is an entire project consists of Planning/Requirements Analysis, Design, Coding, Testing, and Documentation. Various principles of Agile Software Development are:

a) Customer satisfaction by rapid, continuous delivery of useful software.

b) Working software is delivered frequently.

c) Working software is the principle measure of progress.

d) Even late changes in requirements are welcomed.

e) Close, daily cooperation between business people and developers.

f) Face-to-face conversation is the best form of communication.

g) Projects are built around motivated individuals, who should be trusted.

h) Continuous attention to technical excellence and good design.

i) Simplicity.

j) Self-organizing teams.

k) Regular adaptation to changing circumstances.

Various problems with Agile Software Development and their general solutions are given below:

Problem: Schedule Slips

Solution: Short release cycles (a few months at most).

Problem: Project Canceled

Solution: Asks the customer to choose the smallest release that makes the most business sense.

Problem: System goes sour

Solution: Creates and maintains a comprehensive suite of tests which are run and re-run after every change (several times a day) to ensure a quality baseline.

Problem: Defect Rate

Solution: Programmer-written tests(function-by-function) and customer-written tests(feature-by-feature).

Problem: Business Misunderstood

Solution: Call the customer to be an integral part of the team

Problem: Business Changes

Solution: Shorten the release cycle.

Problem: False feature rich

Solution: Insists that only the highest priority tasks are addressed

Problem: Staff Turnover

Solution: Asks to respect programmers to accept responsibility for development.

To solve all these problems in extreme manner, there was Extreme Programming.

3. WHY EXTREME PROGRAMMING?

Traditional process called the Waterfall Methodology used the following steps:

- Decide what to build (requirements and analysis).
- Decide how to build it (design).
- Build it (coding).
- Test it (testing).

Consequences of using traditional S/W development methodology are discussed below:

- No user feedback until very late in the project.
- Requirements changes expensive.
- Progress unmeasurable until late in the project.
- Usually late, so have to make up time somewhere towards the end.
- Testing always seems to get the short stick.
- No deliverable value until end of project.
- Ugly Cost of Change Curve.

The strong need for extreme programming can be described by the use of following points:

a) Extreme Programming (XP) was created in response to problem domains whose requirements change. Customers may not have a firm idea of what the system should do. One may have a system whose functionality is expected to change every few months.. This is when XP will succeed while other methodologies don't. Extreme Programming empowers your developers to confidently respond to changing customer requirements, even late in the life cycle.

b) Extreme Programming improves a software project in five essential ways; communication, simplicity, feedback, respect, and courage. Extreme Programmers constantly communicate with their customers and fellow programmers. They keep their design simple and clean. They get feedback by testing their software starting on day one. They deliver the system to the customers as early as possible and implement changes as suggested. Every small success deepens their respect for the unique contributions of each and every team member. With this foundation Extreme Programmers are able to courageously respond to changing requirements and technology.

c) XP was also set up to address the problems of project risk. If the customers need a new system by a specific date, the risk is high. If that system is a new challenge for your software group the risk is even greater. If that system is a new challenge to the entire software industry the risk is greater even still. The XP practices are set up to mitigate the risk and increase the likelihood of success.

d) XP is set up for small groups of programmers. Between 2 and 12, though larger projects of 30 have

reported success. Programmers can be ordinary; you don't need programmers with a Ph.D. to use XP. But you can not use XP on a project with a huge staff. We should note that on projects with dynamic requirements or high risk one may find that a small team of XP programmers will be more effective than a large team anyway.

e) Extreme Programming is successful because it stresses customer satisfaction. Instead of delivering everything you could possibly want on some date far in the future this process delivers the software you need as you need it.

f) Extreme Programming emphasizes teamwork. Managers, customers, and developers are all equal partners in a collaborative team. Extreme Programming implements a simple, yet effective environment enabling teams to become highly productive. The team self-organizes around the problem to solve it as efficiently as possible.

Extreme Programming is one of several popular Agile Processes. It has already been proven to be very successful at many companies of all different sizes and industries world wide.

4. RULES OF XP

The most surprising aspect of Extreme Programming is its simple rules. Extreme Programming is a lot like a jig saw puzzle. There are many small pieces. Individually the pieces make no sense, but when combined together a complete picture can be seen. The rules may seem awkward and perhaps even naive at first, but are based on sound values and principles.

The rules set expectations between team members but are not the end goal themselves. We will come to realize these rules define an environment that promotes team collaboration and empowerment that is your goal. Once achieved productive teamwork will continue even as rules are changed to fit your company's specific needs. The flow chart in fig 1 shows how Extreme Programming's rules work together. Customers enjoy being partners in the software process, developers actively contribute regardless of experience level, and managers concentrate on communication and relationships. Unproductive activities have been trimmed to reduce costs and frustration of everyone involved. XP 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. In Extreme Programming, every contributor to the project is an integral part of the "Whole Team". The team forms around a business representative called "the Customer", who sits with the team and works with them daily.

4.1. Some Common XP Terms

a) Business the part of an organization that wants a program written, usually because they can make or save money by using it themselves, or make money by selling it.

b) Customer a person or group of people who represent the interests of business to the development team. The ideal customer is either a user of the system, or a proxy for the users, such as a product manager.

c) Development the part of an organization that writes programs, usually to meet the needs/requirements of business.

d) Iteration a period of fixed duration (typically 1, 2 or 3 weeks) during which a set of features is added to the system.

e) User Story a description, written by the customer, of a single desired additional feature of the system being developed. Typically written on a 4x6 card. User stories serve the same purpose as use cases but are not the same. They are used to create time estimates for the release planning meeting. They are also used instead of a large requirements document. User Stories are written by the customers as things that the system needs to do for them. They are similar to usage scenarios, except that they are not limited to describing a user interface. They are in the format of about three sentences of text written by the customer in the customers terminology without techno-syntax.

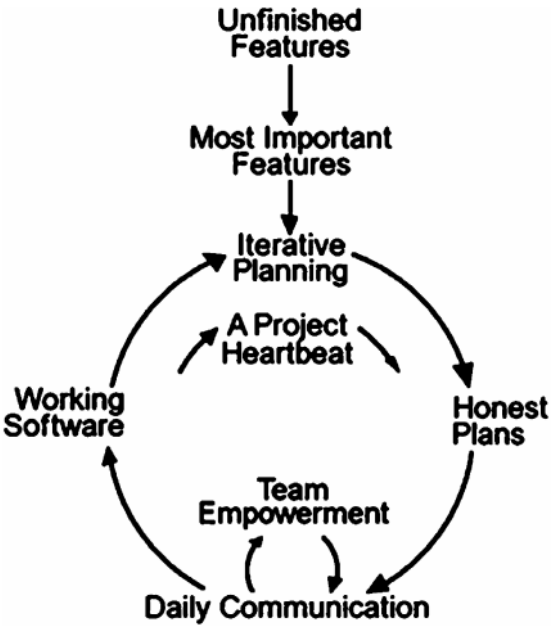

Fig. 1: Rules of Extreme Programming

4.2. Requirements for Extreme Programming

a) An Extended Development Team: The XP team includes not only the developers, but the managers and customers as well, all working together elbow to elbow. Asking questions, negotiating scope and schedules, and creating functional tests require more than just the developers be involved in producing the software.

b) Testability: One must be able to create automated unit and functional tests. While some domains will be disqualified by this requirement, many are not. One need to apply a little testing ingenuity in some domains. System design must be easier to test.

c) Productivity: XP projects unanimously report greater programmer productivity when compared to other projects within the same corporate environment. But this was never a goal of the XP methodology. The real goal has always been to deliver the software that is needed when it is needed. If this is what is important to one's project it may be time to try XP.

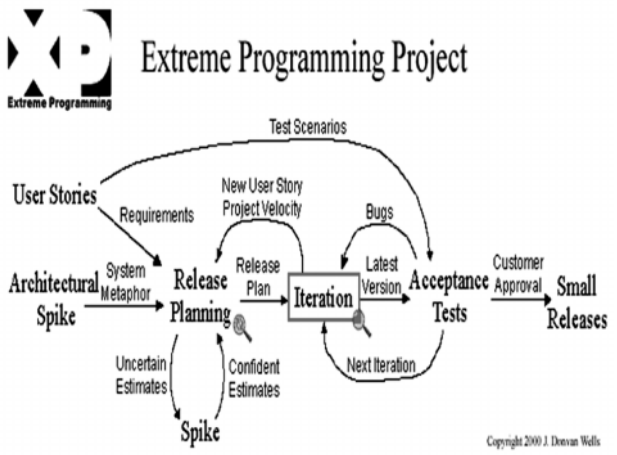

Fig. 2: Typical Stages in XP Project

5. BASIC VALUES OF EXTREME PROGRAMMING

XP uses a consistent set of values that serves both human and commercial needs.

1) Communication: Building software systems requires communicating system requirements to the developers of the system. In formal software development methodologies, this task is accomplished through documentation. Extreme Programming techniques can be viewed as methods for rapidly building and disseminating institutional knowledge 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. To this end, Extreme Programming favors simple designs, common metaphors, collaboration of users and programmers, frequent verbal communication, and feedback.

2) Simplicity: Extreme Programming encourages starting with the simplest solution. Extra functionality can then be added later. The difference between this approach and more conventional system development methods is the focus on designing and coding for the needs of today instead of those of tomorrow, next week, or next month. Proponents of XP acknowledge the disadvantage that this can sometimes entail more effort tomorrow to change the system; their claim is that this is more than compensated for by the advantage of not investing in possible future requirements that might change before they become relevant. Coding and designing for uncertain future requirements implies the risk of spending resources on something that might not be needed. Related to the "communication" value, simplicity in design and coding should improve the quality of communication. A simple design with very simple code could be easily understood by most programmers in the team.

3) Feedback: Within Extreme Programming, feedback relates to different dimensions of the system development:

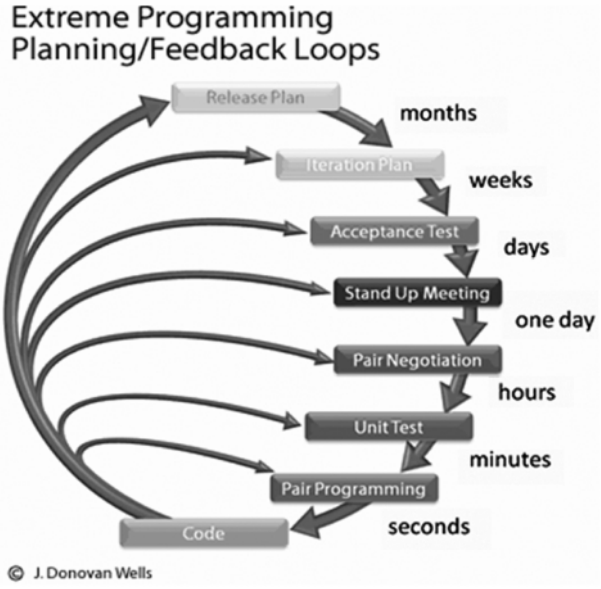

Fig. 3: XP Feedback Loops

- Feedback from the System: By writing unit tests, or running periodic integration tests, the programmers have direct feedback from the state of the system after implementing changes.

- Feedback from the Customer: The functional tests (aka acceptance tests) are written by the customer and the testers. They will get concrete feedback about the current state of their system. This review is planned once in every two or three weeks so the customer can easily steer the development.
- Feedback from the Team: 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.
- Feedback is closely related to communication and simplicity. Flaws in the system are easily communicated by writing a unit test that proves a certain piece of code will break. The direct feedback from the system tells programmers to recode this part. A customer is able to test the system periodically according to the functional requirements, known as user stories.

5. COURAGE

Several practices embody courage. One is the commandment to always design and code for today and not for tomorrow. This is an effort to avoid getting bogged down in design and requiring a lot of effort to implement anything else. 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. Another example of courage is knowing when to throw code away: courage to remove source code that is obsolete, no matter how much effort was used to create that source code. Also, courage means persistence: A programmer might be stuck on a complex problem for an entire day, then solve the problem quickly the next day, if only they are persistent.

6. RESPECT

The respect value manifests in several ways. In Extreme Programming, team members respect each other because programmers should never commit changes that break compilation, that make existing unit-tests fail, or that otherwise delay the work of their peers. Members respect their work by always striving for high quality and seeking for the best design for the solution at hand through refactoring.

Adopting four earlier values led to respect gained from others in the team. Nobody on the team should feel unappreciated or ignored. This ensures high level of motivation and encourages loyalty toward the team, and the goal of the project. This value is very dependent upon the other values, and is very much oriented toward people in a team.

7. XP PRACTICES

XP is extreme in the sense that it takes 12 well-known software development "best practices" to their logical extremes. Extreme Programming (XP) is a new and acclaimed approach to software development process. The goal of XP it to allow software developers to embrace change and successfully address the problem of vague and changing requirements. XP is one of several agile processes, which distinguish themselves from more traditional approaches by their innovative lightweight techniques. XP involves using a number of explicit practices, some of which address technical issues, such as code refactoring and test development, and some of which address human issues, such the on-site customer, and the planning game.

1. Planning Process

The XP planning process allows the XP "customer" to define the business value of desired features, and uses cost estimates provided by the programmers, to choose what needs to be done and what needs to be deferred. The effect less hard-copy documentation - often one of the most of XP's planning process is that it is easy to steer the expensive parts of a software project. The planning process is divided into two parts:

- Release Planning: This is focused on determining what requirements are included in which near-term releases, and when they should be delivered. The customers and developers are both part of this. Release Planning consists of three phases:
  - — Exploration Phase: In this phase the customer will provide a short list of high-value requirements for the system. These will be written down on user story cards.
  - — Commitment Phase: Within the commitment phase business and developers will commit themselves to the functionality that will be included and the date of the next release.
  - — Steering Phase: In the steering phase the plan can be adjusted, new requirements can be added and/or existing requirements can be changed or removed.
- Iteration Planning: This plans the activities and tasks of the developers. In this process the customer is not involved. Iteration Planning also consists of three phases:
  - — Exploration Phase: Within this phase the requirement will be translated to different tasks. The tasks are recorded on task cards.
  - — Commitment Phase: The tasks will be assigned to the programmers and the time it takes to complete will be estimated.

— Steering Phase: The tasks are performed and the end result is matched with the original user story.

2. Small Releases

XP teams put a simple system into production early, and update it frequently on a very short cycle. The delivery of the software is done in predetermined releases (sometimes called 'Builds'). The release plan is determined when initiating the project. Usually each release will carry a small segment of the total software, which can run without depending on components that will be built in the future. The small releases help the customer to gain confidence in the progress of the project.

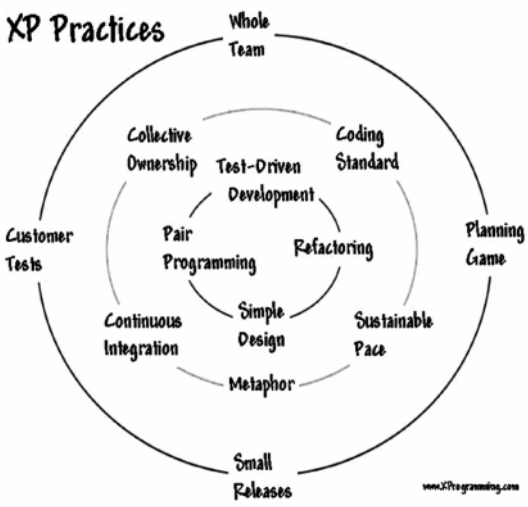

Fig. 4: Extreme Programming Practices

3. Metaphors

XP teams use a common "system of names" and a common system description that guides development and communication. At one level, metaphor and architecture are synonyms they are both intended to provide a broad view of the projects goal. But architectures often get bogged down in symbols and connections. XP uses metaphor in an attempt to define an overall coherent theme to which both developers and business clients can relate.

4. Simple Design

A program built with XP should be the simplest program that meets the current requirements. There is not much building "for the future". Instead, the focus is on providing business value. Of course it is necessary to ensure that you have a good design, and in XP this is brought about through "refactoring". Simple design has two parts: 1) design for the functionality that has been defined, not for potential future functionality. 2) create the best design that can deliver that functionality.

5. Continuous Testing

Before programmers add a feature, they write a test for it. When the suite runs, the job is done. Tests in XP come in two basic flavors.

a. Unit Tests are automated tests written by the developers to test functionality as they write it. Each unit test typically tests only a single class, or a small cluster of classes. Unit tests are typically written using a unit testing framework, such as JUnit.

b. Acceptance Tests (also known as Functional Tests) are specified by the customer to test that the overall system is functioning as specified. Acceptance tests typically test the entire system, or some large chunk of it. When all the acceptance tests pass for a given user story, that story is considered complete. At the very least, an acceptance test could consist of a script of user interface actions and expected results that a human can run. Ideally acceptance tests should be automated, either using the unit testing framework, or a separate acceptance testing framework.

6. Refactoring

One thing that sets XP apart from other approaches, it would be refactoring the ongoing redesign of software to improve its responsiveness to change. The refactoring process focuses on removal of duplication (a sure sign of poor design), and on increasing the "cohesion" of the code, while lowering the "coupling". High cohesion and low coupling have been recognized as the hallmarks of well-designed code for at least thirty years.

7. Pair Programming

XP programmers write all production code in pairs, two programmers working together at one machine. Pair programming has been shown by many experiments to produce better software at similar or lower cost than programmers working alone.

8. Collective Ownership

On an Extreme Programming project, any pair of programmers can improve any code at any time. This means that all code gets the benefit of many people's attention, which increases code quality and reduces defects.

9. Continuous Integration

XP teams integrate and build the software system multiple times per day. This keeps all the programmers on the same page, and enables very rapid progress. Perhaps surprisingly, integrating more frequently tends to eliminate integration

problems that plague teams who integrate less often. Extreme Programming teams keep the system fully integrated at all times.

10. 40-hour Week

Tired programmers make more mistakes. XP teams do not work excessive overtime, keeping them fresh, healthy, and effective. Extreme Programming teams are in it for the long term. They work hard, and at a pace that can be sustained indefinitely. This means that they work overtime when it is effective. XP teams are in it to win, not to die.

11. On-site Customer

An XP project is steered by a dedicated individual who is empowered to determine requirements, set priorities, and answer questions as the programmers have them. The effect of being there is that communication improves, with less hard-copy documentation - often one of the most expensive parts of a software project.

12. Coding Standard

For a team to work effectively in pairs, and to share ownership of all the code, all the programmers need to write the code in the same way, with rules that make sure the code communicates clearly. XP teams follow a common coding standard, so that all the code in the system looks as if it was written by a single — very competent — individual.

8. CONCLUSION

Extreme Programming 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. XP doesn't ask developers to abandon good software engineering practices. It does, however, asks to consider closely the absolute minimum set of practices that enable a small, co-located team to function effectively in today's software delivery development.

XP is a good choice when requirements are unclear, or prone to change (because of changing business situations, or as a result of external conditions). XP is distinguished from others by:

- Early, concrete, and continuing feedback from short life cycles.
- Incremental planning approach.
- Ability to flexibly schedule the implementation of functionality, responding to changing business needs.
- Reliance on automated tests written by programmers and customers.
- Reliance on oral communication, tests, and source code to communicate system structure and intent.
- Reliance on evolutionary design process that lasts as long as the system lasts.
- Reliance on the close collaboration of programmers with ordinary skills.
- Reliance on practices that work with both the short term instincts of programmers and long term interests of the project.

REFERENCES

[1] Doo Hwan Bae, "Extreme Programming".

[2] Lisa Crispin, Tip House, Carol Wade, "The Need for Speed: Automating Acceptance Testing in an eXtreme Programming Enviornment"

[3] Michael McCormick, "Programming Extremism", Communications of ACM, 44, No. 6, June 2001, pp 199-201.

[4] Cesar F. Acebal, Juan M. Cueva Lovelle, "A New Method of Software Development: Extreme Programming."

[5] K.Beck, Extreme Programming Explained. Embrace Change, Addison –Wesley,2000.

[6] http://thoughtworks.com

[7] Jeffries, R. http://www.xprogramming.com

[8] http://groups.yahoo.com/group/xpstl