Agile – Extreme Programming

By
Sunil Kumar(Master of Sc.)
Bangalore, India

Agenda

- History of XP
- Overview of XP
- Activities
- Value
- Principles
- Practices
- Rules
  - Planning
  - Managing
  - Design
  - Coding
  - Testing
- Summary

History of Extreme Programming(XP)

- Extreme Programming is a software development methodology which is intended to improve software quality and responsiveness to changing customer requirements.
- Extreme Programming takes its name from the idea that the beneficial elements of traditional software engineering practices are taken to extreme levels.
  - As an example, code reviews are considered a beneficial practice. Taken to the extreme, code can be reviewed continuously with the practice of pair programming.

History of Extreme Programming(XP)

- Extreme Programming was created by Kent Beck during his work of Comprehensive Compensation System payroll project(C3).
- In 1996, Chrysler called in Kent Beck to help with their struggling C3 project as an external consultant.
- Initially, Chrysler attempted to implement a package solution, but it failed because of the complexity surrounding the rules and integration. From this point of crisis, Kent Beck and his team took over, effectively starting the project from scratch. The classic Waterfall development approach had been tried and failed, so something drastic was required.
- Fundamentally, the C3 team focused on the business value the customer wanted, and discarded anything that did not work towards that goal.
- Extreme Programming was created by developers for developers. The XP team at Chrysler was able to deliver their first working system within 1 year. Development continued over the next year with new functionality being added through smaller releases.

Overview of Extreme Programming

- Extreme Programming can be described as a software development discipline that organizes people to produce high-quality software more productively.
- Extreme Programming attempts to reduce the cost of changing requirements by having multiple short development cycles rather than one long one like in Waterfall.
- With Extreme Programming, changes are a natural, inescapable, and desirable aspect of software development projects, and should be planned for instead of attempting to define a stable set of requirements up front.

Overview of Extreme Programming

- 4x Activities
  - Coding, testing, listening, and designing.
- 5x Values
  - Communication, simplicity, feedback, courage, and respect
- 3x Principles
  - Feedback, assuming simplicity, and embracing change.
- 12x Practices under four groups
  - Fine-scale feedback, continuous process, shared understanding, and programmer welfare.
- 29x Rules split into the five groups.
  - Planning, managing, designing, coding, and testing. Let's first take a look at the four activities.

4- Activities

- Coding
  - Coding is the most important product of the Extreme Programming process. Without code, there is no working product.
- Testing
  - We cannot be certain of having a working system or product unless we have tested it. With Extreme Programming, we ideally want to automate as much of your testing as possible so that you can repeat the testing frequently. This is done by writing unit tests cases.
  - With XP, the developer will practice test-driven development. We write a failing test first and implement just enough code to pass the test, and then refactor the code to a better structure, while tests still pass.
  - The programmer will strive to cover as much of their code in unit tests as they can to give them a good level of overall code coverage. This code coverage will help build up trust that the system operates as expected.

4- Activities

- Listening
  - Programmers must listen to what the customers need the system to do and what business logic is required.
  - They must understand these needs well enough to give the customer feedback about the technical aspects of how the problem can be solved or cannot be solved.
  - The requirements from the customer are documented as a series of user stories. These user stories help to drive out a series of acceptance tests, which help determine when a user story is completed and working as expected. Once user stories and acceptance tests are written, the developers can then start their planning and estimating.

4- Activities

- Designing
  - To create a working system or product, requirement gathering, coding, and testing should be all you need, but in reality software systems are very complicated, so you will need to perform a level of overall system design.
  - This doesn't mean that you would need to create a several hundred page design document, as that could be quite wasteful, but there is definite value in producing an overall system design where you look at the overall structure of the system and its dependencies.
  - Ideally you want to create a system where all of the components are as decoupled from each other as they can be, so that a change in one component doesn't require sweeping changes across the rest of the system.

5-Values

- Communication
  - Good communication is essential to any project. Honest, regular communication allows you to adjust to change. XP puts developers and customers in constant communication. A customer set business priorities.
  - When we have a question about a feature, we should ask the customer directly. A 5 minute face-to-face conversation, peppered with body language, gestures, and white board drawings, communicates more than an email exchange or conference call can, so removing the communication barriers between customers and developers increases your flexibility.

5-Values

- Build for Simplicity
  - Simplicity means building only the system that really needs to be built. It means solving only today's problems today.
  - Complexity costs a lot and predicting the future is very hard. Armed with communication and feedback it's much easier to know exactly what you need.

5-Values

- Learning from Feedback
  - Feedback means asking questions and learning from the answers. The sooner we can get feedback, the more time we have to react to it.
  - XP provides rapid, frequent feedback. Every XP practice is part of building feedback loop. The best way to reduce the cost of change is to listen to and learn from all of those sources as often as possible. This is why XP concentrates on frequent planning, design, testing, and communicating.
  - Rapid feedback reduces the investment of time and resources in ideas with little payoff. Failures are found as soon as possible, within days or weeks rather than months or years, and this feedback helps you to refine your schedule and your plans.
  - It allows you to steer your project back on track as soon as someone notices a problem and identifies when a feature is finished.

5-Values

- Having Courage
  - Courage means making the hard decisions when necessary. If a feature isn't working, fix it. If some code is not up to standard, improve it. If you're not going to deliver everything you promised on schedule, be up front and tell the customer as soon as possible.
  - Courage is a difficult virtue to apply. No one wants to be wrong or to break a promise.
  - The only way to recover from a mistake, though, is to admit it and fix it. Delivering software is challenging, but meeting that challenge instead of avoiding it, leads to better software.

5-Values

- Having Respect for the team and Project
  - Respect underlies the other values previously mentioned. Every member of the team must care about the project.
  - Intrinsic rewards like motivation, enjoyment, and job satisfaction beat extrinsic reward like employee-of-the-month awards or physical rewards every time.
  - Developers should always respect the expertise of the customers and vice-versa, and managers should always respect the developers right to accept responsibility and receive authority over their work.

3-Principles

- Feedback
  - XP stresses that minimal delay between an action and its feedback is critical to learning and making changes. With frequent feedback from the customer, a mistaken design decision made by the developer will be noticed and corrected quickly, before the developer spends much time implementing it.
  - Unit tests contribute greatly to the rapid feedback principle. This includes running not only the unit tests that test the developer's code, but running, in addition, all unit tests against the software using an automated process that can be initiated by a single command as part of a build.

3-Principles

- Assuming Simplicity
  - Assuming simplicity is about treating every problem as if its solution were extremely simple.
  - Traditional system development methods say to plan for the future and to code for reusability. Extreme Programming rejects these ideas.
  - Extreme Programming applies incremental changes. For example, a system might have small releases every 3 weeks. When many little steps are made, the customer has more control over the development process and the system that is being developed.

3-Principles

- Embracing Changes
  - The principle of embracing change is about not working against changes, but embracing them.
  - For instance, if at one of the iteration planning meetings it appears the customer's requirements have changed dramatically, programmers are to embrace this and plan new requirements for the next iteration.
  - Under Waterfall, changes in requirements are seen as a very bad and costly thing to happen. Even small changes can have a very large impact to a program at work. If any of the main fundamental requirements change under Waterfall, it could put the entire project at risk of being cancelled. This risk is drastically minimized.