Extreme Programming - Introduction

This chapter gives an overview of Extreme Programming.

What is Agile?

The word ‘agile’ means −

- Able to move your body quickly and easily.
- Able to think quickly and clearly.

In business, ‘agile’ is used for describing ways of planning and doing work wherein it is understood that making changes as needed is an important part of the job. Business‘agililty’ means that a company is always in a position to take account of the market changes.

Ref: Cambridge Dictionaries online.

In software development, the term ‘agile’ is adapted to mean ‘the ability to respond to changes − changes from Requirements, Technology and People.’

Agile Manifesto

A team of software developers published the Agile Manifesto in 2001, highlighting the importance of the development team, accommodating changing requirements and customer involvement.

The Agile Manifesto states that −

We are uncovering better ways of developing software by doing it and helping others do it. Through this work, we have come to value −

- **Individuals and interactions** over processes and tools.
- **Working software** over comprehensive documentation.
- **Customer collaboration** over contract negotiation.
- **Responding to change** over following a plan.

That is, while there is value in the items on the right, we value the items on the left more.

Characteristics of Agility

Following are the characteristics of Agility −

- Agility in Agile Software Development focuses on the culture of the whole team with multi-discipline, cross-functional teams that are empowered and selforganizing.
- It fosters shared responsibility and accountability.
- Facilitates effective communication and continuous collaboration.
- The whole-team approach avoids delays and wait times.
- Frequent and continuous deliveries ensure quick feedback that in in turn enable the team align to the requirements.
- Collaboration facilitates combining different perspectives timely in implementation, defect fixes and accommodating changes.
- Progress is constant, sustainable, and predictable emphasizing transparency.

Software Engineering Trends

The following trends are observed in software engineering −

- Gather requirements before development starts. However, if the requirements are to be changed later, then following is usually noticed −
    - Resistance to the changes at a later stage of development.
    - There is a requirement of a rigorous change process that involves a change control board that may even push the changes to later releases.
    - The delivery of a product with obsolete requirements, not meeting the customer's expectations.
    - Inability to accommodate the inevitable domain changes and technology changes within the budget.
- Find and eliminate defects early in the development life cycle in order to cut the defect-fix costs.
    - Testing starts only after coding is complete and testing is considered as a tester's responsibility though the tester is not involved in development.
    - Measure and track the process itself. This becomes expensive because of −
    - Monitoring and tracking at the task level and at the resource level.
    - Defining measurements to guide the development and measuring every activity in the development.
    - Management intervention.

- Elaborate, analyze, and verify the models before development.
    - A model is supposed to be used as a framework. However, focus on the model and not on the development that is crucial will not yield the expected results.
- Coding, which is the heart of development is not given enough emphasis. The reasons being −
    - Developers, who are responsible for the production, are usually not in constant communication with the customers.
    - Coding is viewed as a translation of design and the effective implementation in code is hardly ever looped back into the design.
- Testing is considered to be the gateway to check for defects before delivery.
    - Schedule overruns of the earlier stages of development are compensated by overlooking the test requirements to ensure timely deliveries.
    - This results in cost overruns fixing defects after delivery.
    - Testers are made responsible and accountable for the product quality though they were not involved during the entire course of development.
- Limiting resources (mainly team) to accommodate budget leads to −
    - Resource over allocation
    - Team burnout.
    - Loss in effective utilization of team competencies.
    - Attrition.

Extreme Programming − A way to handle the common shortcomings

Software Engineering involves −

- Creativity
- Learning and improving through trials and errors
- Iterations

Extreme Programming builds on these activities and coding. It is the detailed (not the only) design activity with multiple tight feedback loops through effective implementation, testing and refactor

continuously.

Extreme Programming is based on the following values −

- Communication
- Simplicity
- Feedback
- Courage
- Respect

What is Extreme Programming?

XP is a lightweight, efficient, low-risk, flexible, predictable, scientific, and fun way to develop a software.

e**X**treme **P**rogramming (XP) was conceived and developed to address the specific needs of software development by small teams in the face of vague and changing requirements.

Extreme Programming is one of the Agile software development methodologies. It provides values and principles to guide the team behavior. The team is expected to self-organize. Extreme Programming provides specific core practices where −

- Each practice is simple and self-complete.
- Combination of practices produces more complex and emergent behavior.

Embrace Change

A key assumption of Extreme Programming is that the cost of changing a program can be held mostly constant over time.

This can be achieved with −

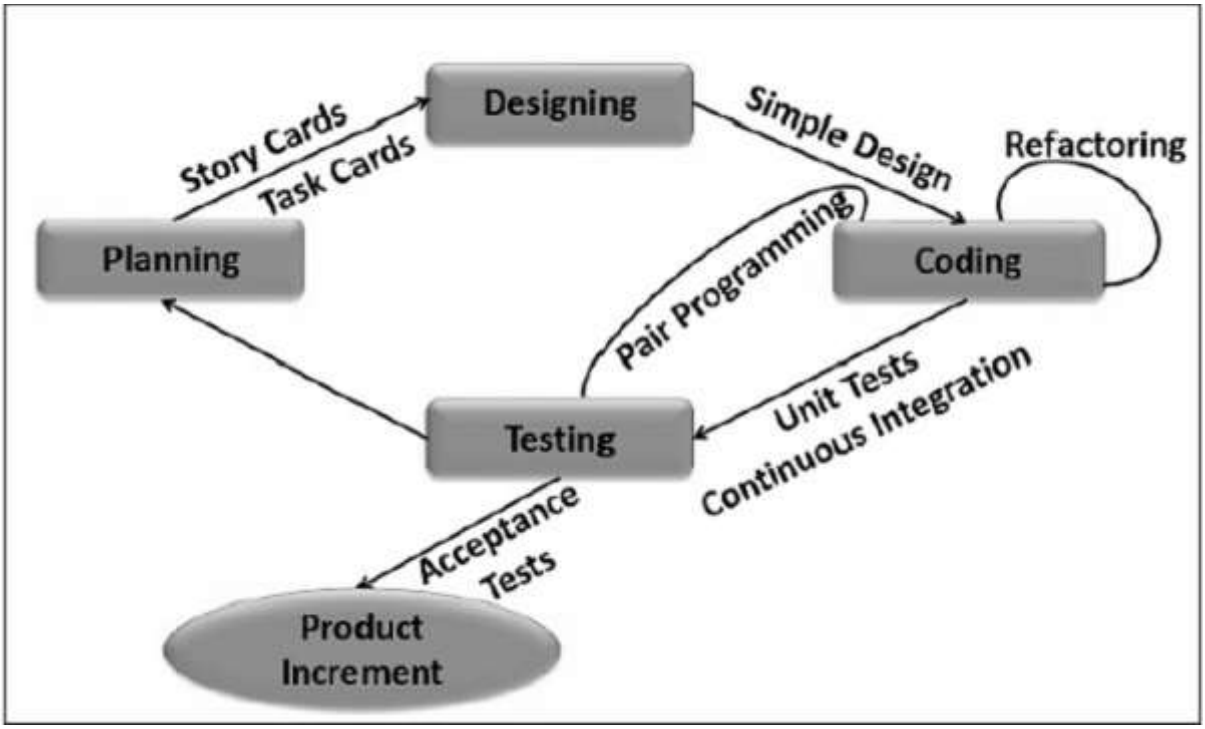
- Emphasis on continuous feedback from the customer
- Short iterations
- Design and redesign
- Coding and testing frequently
- Eliminating defects early, thus reducing costs
- Keeping the customer involved throughout the development
- Delivering working product to the customer

Extreme Programming in a Nutshell

Extreme Programming involves −

- Writing unit tests before programming and keeping all of the tests running at all times. The unit tests are automated and eliminates defects early, thus reducing the costs.
- Starting with a simple design just enough to code the features at hand and redesigning when required.
- Programming in pairs (called pair programming), with two programmers at one screen, taking turns to use the keyboard. While one of them is at the keyboard, the other constantly reviews and provides inputs.
- Integrating and testing the whole system several times a day.
- Putting a minimal working system into the production quickly and upgrading it whenever required.
- Keeping the customer involved all the time and obtaining constant feedback.

Iterating facilitates the accommodating changes as the software evolves with the changing requirements.

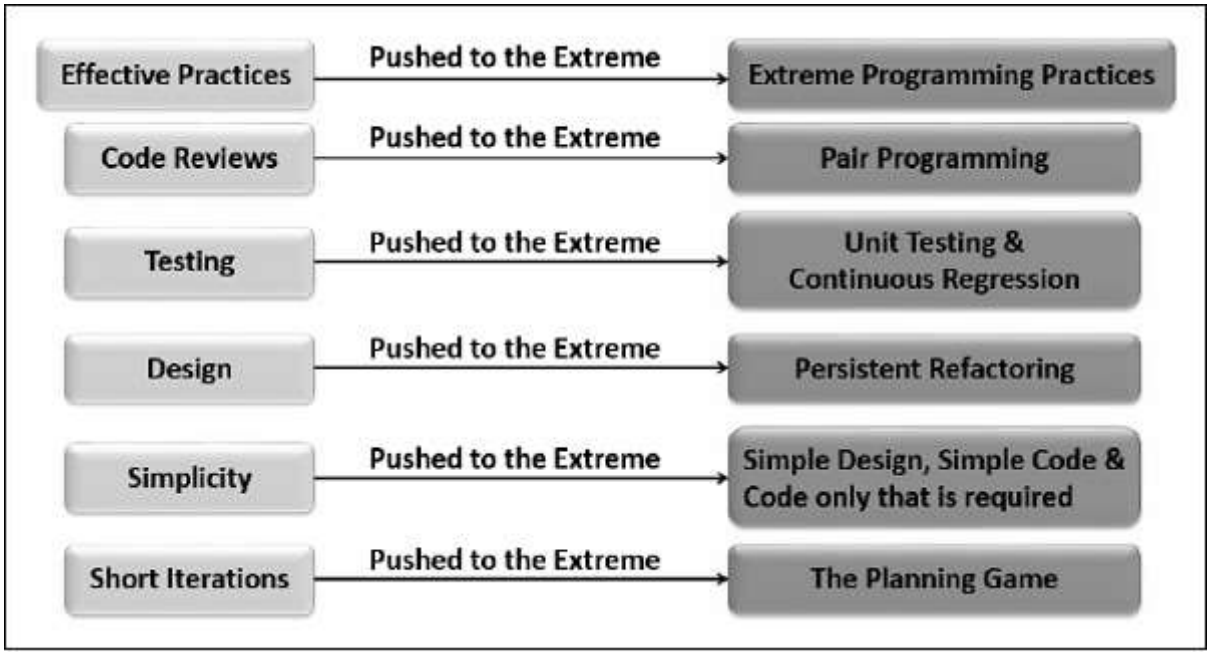


Why is it called “Extreme?”

Extreme Programming takes the effective principles and practices to extreme levels.

- Code reviews are effective as the code is reviewed all the time.

- Testing is effective as there is continuous regression and testing.
- Design is effective as everybody needs to do refactoring daily.
- Integration testing is important as integrate and test several times a day.
- Short iterations are effective as the planning game for release planning and iteration planning.



History of Extreme Programming

Kent Beck, Ward Cunningham and Ron Jeffries formulated extreme Programming in 1999. The other contributors are Robert Martin and Martin Fowler.

In Mid-80s, Kent Beck and Ward Cunningham initiated Pair Programming at Tektronix. In the 80s and 90s, Smalltalk Culture produced Refactoring, Continuous Integration, constant testing, and close customer involvement. This culture was later generalized to the other environments.

In the Early 90s, Core Values were developed within the Patterns Community, Hillside Group. In 1995, Kent summarized these in Smalltalk Best Practices, and in 1996, Ward summarized it in episodes.

In 1996, Kent added unit testing and metaphor at Hewitt. In 1996, Kent had taken the Chrysler C3 project, to which Ron Jeffries was added as a coach. The practices were refined on C3 and published on Wiki.

Scrum practices were incorporated and adapted as the planning game. In 1999, Kent published his book, ‘Extreme Programming Explained’. In the same year, Fowler published his book, Refactoring.

Extreme Programming has been evolving since then, and the evolution continues through today.

Success in Industry

The success of projects, which follow Extreme Programming practices, is due to −

- Rapid development.
- Immediate responsiveness to the customer's changing requirements.
- Focus on low defect rates.
- System returning constant and consistent value to the customer.
- High customer satisfaction.
- Reduced costs.
- Team cohesion and employee satisfaction.

Extreme Programming Advantages

Extreme Programming solves the following problems often faced in the software development projects −

- **Slipped schedules** − and achievable development cycles ensure timely deliveries.
- **Cancelled projects** − Focus on continuous customer involvement ensures transparency with the customer and immediate resolution of any issues.
- **Costs incurred in changes** − Extensive and ongoing testing makes sure the changes do not break the existing functionality. A running working system always ensures sufficient time for accommodating changes such that the current operations are not affected.
- **Production and post-delivery defects**: **Emphasis is on** − the unit tests to detect and fix the defects early.
- **Misunderstanding the business and/or domain** − Making the customer a part of the team ensures constant communication and clarifications.
- **Business changes −** Changes are considered to be inevitable and are accommodated at any point of time.
- **Staff turnover** − Intensive team collaboration ensures enthusiasm and good will. Cohesion of multi-disciplines fosters the team spirit.