XP

Extreme Programming

tutorialspoint

SIMPLY EASY LEARNING

About the Tutorial

Extreme programming (XP) is a software development methodology, which is intended to improve software quality and responsiveness to changing customer requirements. As a type of agile software development, it advocates frequent "releases" in short development cycles, to improve productivity and introduce checkpoints at which new customer requirements can be adopted.

Audience

XP is a lightweight, efficient, low-risk, flexible, predictable, scientific, and fun way to develop a software. Extreme Programming 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 perceived to be effective in smaller teams, with a team size up to 12-16 developers.

Prerequisites

Before you start proceeding with this tutorial, we are assuming that you are already aware about the basics of Agile methodologies and Scrum. If you are not well aware of these concepts, then we will suggest you to go through our short tutorials on Agile and Scrum.

Copyright & Disclaimer

Table of Contents

1. 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 self-organizing.

- 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 refactoring 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-

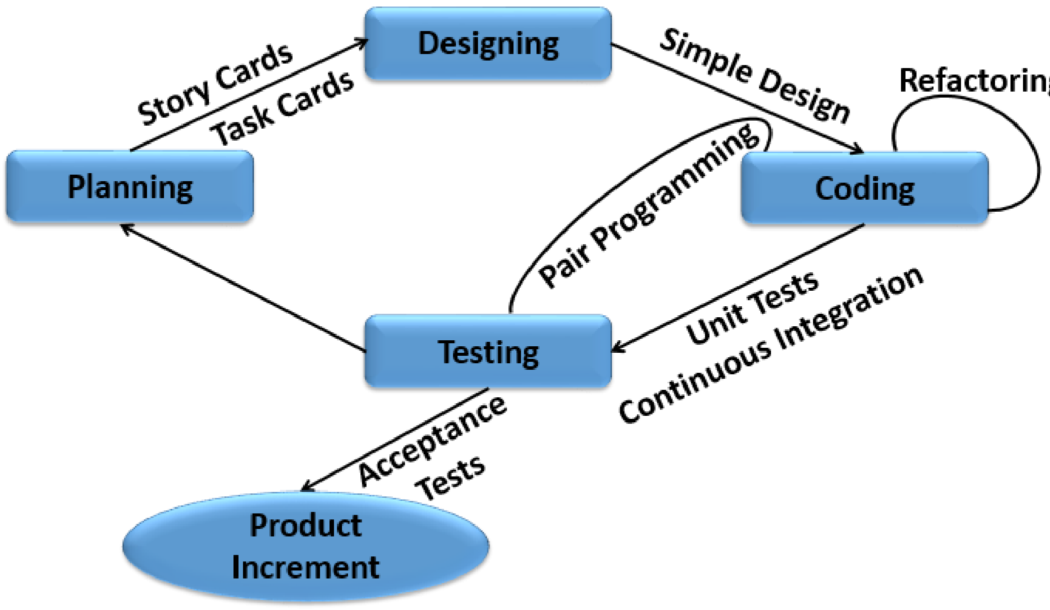
- Emphasis on continuous feedback from the customer
- Short iterations
- Design and redesign
- Coding and testing frequently
- Eliminating defects early, thus reducing costs
- Keeping the customer involved throughout the development
- Delivering working product to the customer

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.

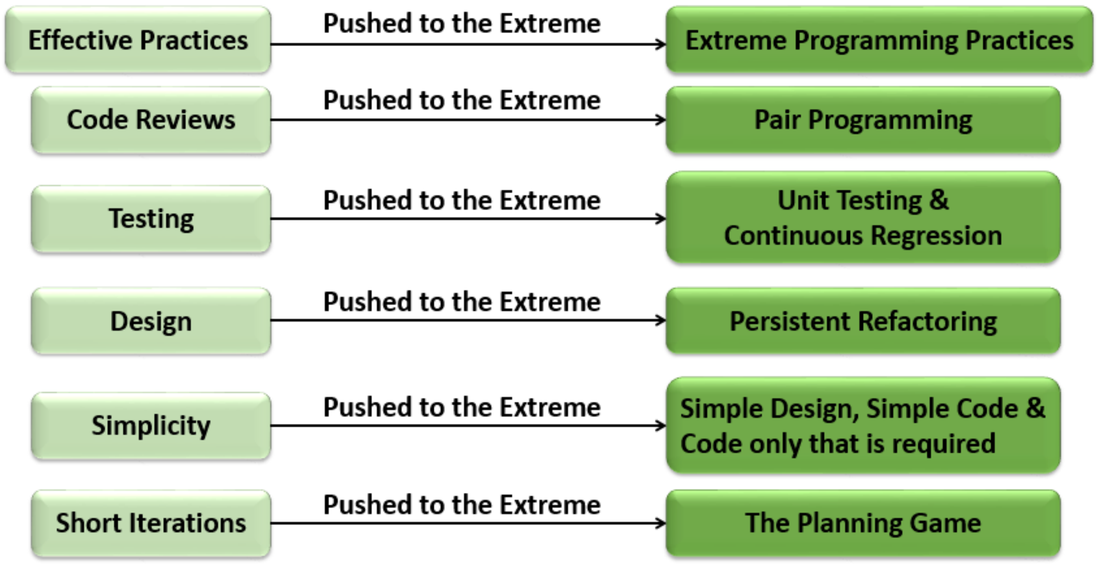
Iterating facilitates the accommodating changes as the software evolves with the changing requirements.



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:** Short 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.

2. Extreme Programming – Values and Principles

XP sets out to lower the cost of change by introducing basic values, principles and practices. By applying XP, a system development project should be more flexible with respect to changes.

Extreme Programming Values

Extreme Programming (XP) is based on the five values-

- Communication
- Simplicity
- Feedback
- Courage
- Respect

Communication

Communication plays a major role in the success of a project. Problems with projects often arise due to lack of communication. Many circumstances may lead to the breakdown in communication. Some of the common problems are-

- A developer may not tell someone else about a critical change in the design.
- A developer may not ask the customer the right questions, and so a critical domain decision is blown.
- A manager may not ask a developer the right question, and project progress is misreported.
- A developer may ignore something important conveyed by the customer.

Extreme Programming emphasizes continuous and constant communication among the team members, managers and the customer. The Extreme Programming practices, such as unit testing, pair programming, simple designs, common metaphors, collective ownership and customer feedback focus on the value of communication.

XP employs a coach whose job is to notice when the people are not communicating and reintroduce them. Face-to-Face communication is preferred and is achieved with pair programming and a customer representative is always onsite.

Simplicity

Extreme Programming believes in 'it is better to do a simple thing today and pay a little more tomorrow to change it' than 'to do a more complicated thing today that may never be used anyway'.

- Do what is needed and asked for, but no more.
    - "Do the simplest thing that could possibly work" The DTSTTCPW principle.
    - Implement a new capability in the simplest possible way. Also known as the KISS principle 'Keep It Simple, Stupid!'.
    - A coach may say DTSTTCPW when he sees an Extreme Programming developer doing something needlessly complicated.
    - Refactor the system to be the simplest possible code with the current feature set. This will maximize the value created for the investment made till date.
-
- Take small simple steps to your goal and mitigate failures as they happen.
- Create something that you are proud of and maintain it for a long term for reasonable costs.
- Never implement a feature you do not need now i.e. the 'You Aren't Going to Need It' (YAGNI) principle.

Communication and Simplicity support each other.

- The more you communicate the clearer you can see exactly what needs to be done, and you gain more confidence about what really need not be done.
- The simpler your system is, the less you have to communicate about the fewer developers that you require. This leads to better communication.

Feedback

Every iteration commitment is taken seriously by delivering a working software. The software is delivered early to the customer and a feedback is taken so that necessary changes can be made if needed. Concrete feedback about the current state of the system is priceless. The value of the feedback is a continuously running system that delivers information about itself in a reliable way.

In Extreme Programming, feedback is ensured at all levels at different time scales-

- Customers tell the developers what features they are interested in so that the developers can focus only on those features.

- Unit tests tell the developers the status of the system.
- The system and the code provides feedback on the state of development to the managers, stakeholders and the customers.
- Frequent releases enable the customer to perform acceptance tests and provide feedback and developers to work based on that feedback.
- When the customers write new features/user stories, the developers estimate the time required to deliver the changes, to set the expectations with the customer and managers.

Thus, in Extreme Programming the feedback-

- Works as a catalyst for change
- Indicates progress
- Gives confidence to the developers that they are on the right track

Courage

Extreme Programming provides courage to the developers in the following way-

- To focus on only what is required
- To communicate and accept feedback
- To tell the truth about progress and estimates
- To refactor the code
- To adapt to changes whenever they happen
- To throw the code away (prototypes)

This is possible as no one is working alone and the coach guides the team continuously.

Respect

Respect is a deep value, one that lies below the surface of the other four values. In Extreme Programming,

- Everyone respects each other as a valued team member.
- Everyone contributes value such as enthusiasm.
- Developers respect the expertise of the customers and vice versa.
- Management respects the right of the developers to accept the responsibility and receive authority over their own work.

Combined with communication, simplicity, and concrete feedback, courage becomes extremely valuable.

- Communication supports courage because it opens the possibility for more high-risk, high-reward experiments.

- Simplicity supports courage because you can afford to be much more courageous with a simple system. You are much less likely to break it unknowingly.

- Courage supports simplicity because as soon as you see the possibility of simplifying the system you try it.

- Concrete feedback supports courage because you feel much safer trying radical modifications to the code, if you can see the tests turn green at the end. If any of the tests do not turn green, you know that you can throw the code away.

Extreme Programming Principles

The values are important, but they are vague, in the sense that it may not be possible to decide if something is valuable. For example, something that is simple from someone's point of view may be complex from someone else's point of view.

Hence, in Extreme Programming, the basic principles are derived from the values so that the development practices can be checked against these principles. Each principle embodies the values and is more concrete, i.e. rapid feedback – you either, have it or you do not.

The fundamental principles of Extreme Programming are-

- Rapid feedback
- Assume simplicity
- Incremental change
- Embracing change
- Quality work

Rapid Feedback

Rapid feedback is to get the feedback, understand it, and put the learning back into the system as quickly as possible.

- The developers design, implement and test the system, and use that feedback in seconds or minutes instead of days, weeks, or months.
-
- The customers review the system to check how best it can contribute, and give feedback in days or weeks instead of months or years.

Assume Simplicity

To assume simplicity is to treat every problem as if it can be solved with simplicity.

Traditionally, you are told to plan for the future, to design for reuse. The result of this approach may turn into ‘what is required today by the customer is not met and what is ultimately delivered may be obsolete and difficult to change.’

‘Assume Simplicity’ means ‘do a good job of solving today's job today and trust your ability to add complexity in the future where you need it.’ In Extreme Programming, you are told to do a good job (tests, refactoring, and communication) focusing on what is important today.

- With good unit tests, you can easily refactor your code to do additional tests.
-
- Follow YAGNI (You Ain’t Gonna Need It).
-
- Follow the DRY(Don’t Repeat Yourself) principle. For example,
  -
  - Do not have multiple copies of identical (or very similar) code.
  -
  - Do not have redundant copies of information.
  - No wastage of time and resources on what may not be necessary.
-

Incremental Change

In any situation, big changes made all at once just do not work. Any problem is solved with a series of the smallest change that makes a difference.

In Extreme Programming, Incremental Change is applied in many ways.

- The design changes a little at a time.
- The plan changes a little at a time.
- The team changes a little at a time.

Even the adoption of Extreme Programming must be taken in little steps.

Embracing Change

The best strategy is the one that preserves the most options while actually solving your most pressing problem.

Quality Work

Everyone likes doing a good job. They try to produce the quality that they are proud of. The team