XP

Extreme Programming

tutorialspoint

SIMPLY EASY LEARNING

www.tutorialspoint.com

https://www.facebook.com/tutorialspointindia

https://twitter.com/tutorialspoint

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. XP – 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-

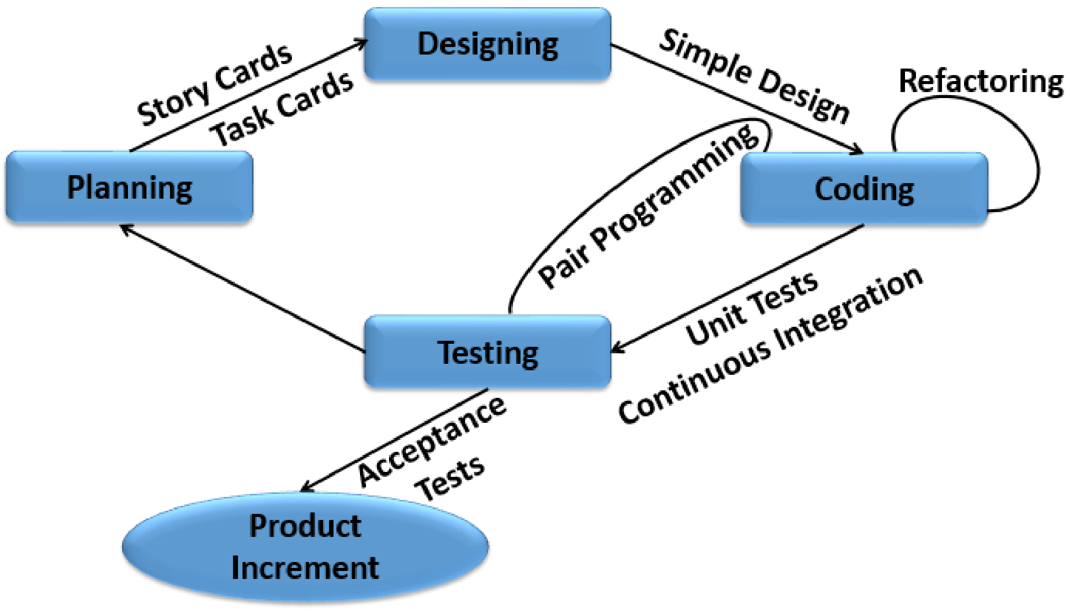
- Writing unit tests before programming and keeping all of the tests running at all times. The unit tests are automated and eliminates defects early, thus reducing the costs.

- 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.

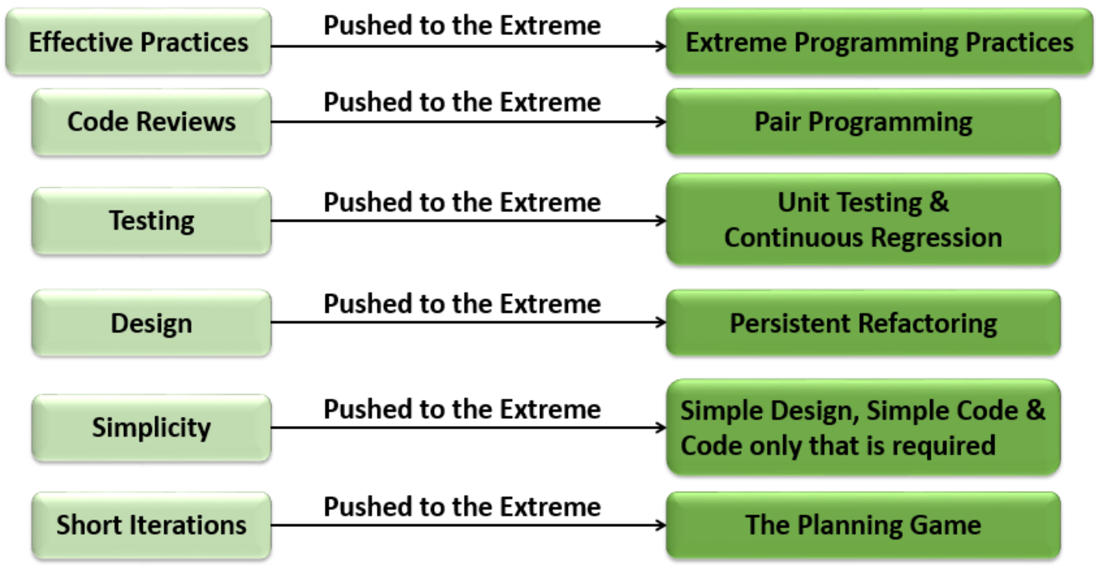
Iterating facilitates the accommodating changes as the software evolves with the changing requirements.



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. XP – 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

- Works well
- Enjoys the work
- Feels good in producing a product of value

3. XP – Practices

There are four basic activities in Extreme Programming. They are-

- Coding
- Testing
- Listening
- Designing

These four basic activities need to be structured in the light of the Extreme Programming principles. To accomplish this, the Extreme Programming practices are defined.

These 12 Extreme Programming practices achieve the Extreme Programming objective and wherever one of the practices is weak, the strengths of the other practices will make up for it.

Kent Beck, the author of 'Extreme Programming Explained' defined 12 Extreme Programming practices as follows-

- The Planning Game
- Short Releases
- Metaphor
- Simple Design
- Testing
- Refactoring
- Pair Programming
- Collective Ownership
- Continuous Integration
- 40 hour Week
- On-site Customer
- Coding Standards

Four Areas of Extreme Programming

The Extreme Programming practices can be grouped into four areas-

- Rapid, Fine Feedback-
    - Testing
    - On-site customer
    - Pair programming

- Continuous Process-
    - Continuous Integration
    - Refactoring
    - Short Releases

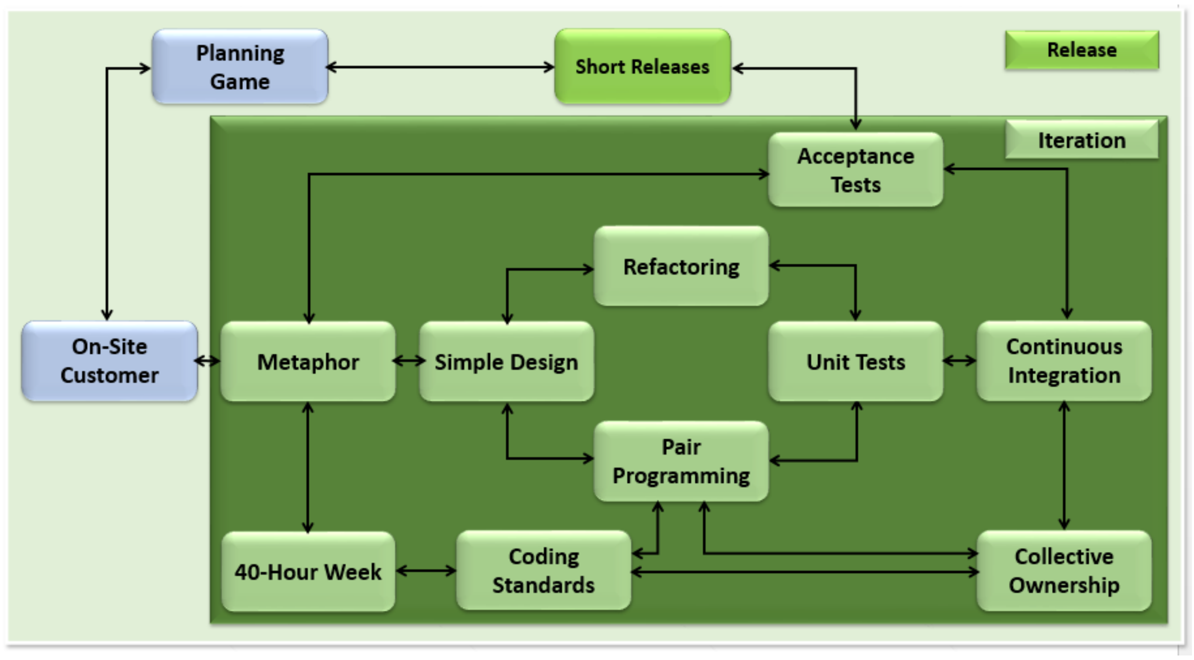
- Shared Understanding-
    - The Planning Game
    - Simple Design
    - Metaphor
    - Collective Ownership
    - Coding Standards

- Developer Welfare-
    - Forty-Hour Week

In this chapter, you will understand the Extreme Programming practices in detail and the advantages of each of these practices.

Extreme Programming Practices at-a-glance

The following diagram shows how Extreme Programming is woven around the Extreme Programming practices-



The Planning Game

The main planning process within extreme programming is called the Planning Game. The game is a meeting that occurs once per iteration, typically once a week. The Planning Game is toqQuickly determine the scope of the next release by combining business priorities and technical estimates. As reality overtakes the plan, update the plan.

Business and development need to make the decisions in tandem. The business decisions and the development's technical decisions have to align with each other.

Business people need to decide about-

- **Scope:** How much of a problem must be solved for the system to be valuable in production? The businessperson is in a position to understand how much is not enough and how much is too much.

- **Priority:** If you are given an option, which one do you want? The businessperson is in a position to determine this, more than a developer with inputs from the customer.

- **Composition of releases:** How much or how little needs to be done before the business is better off with the software than without it? The developer's intuition about this question can be wildly wrong.

- **Dates of releases:** What are important dates at which the presence of the software (or some of the software) would make a big difference?

Technical people need to decide about-

- **Estimates:** How long will a feature take to implement?

- **Consequences**: There are strategic business decisions that should be made only when informed about the technical consequences. Development needs to explain the consequences.

- **Process**: How will the work and the team be organized? The team needs to fit the culture in which it will operate. The software must be written well rather than preserve the irrationality of an enclosing culture.

- **Detailed Scheduling**: Within a release, which stories should be done first? The developers need the freedom to schedule the riskiest segments of development first, to reduce the overall risk of the project. Within that constraint, they still tend to move business priorities earlier in the development, reducing the chance that important stories will have to be dropped towards the end of the development of a release due to time constraints.

Thus, plan is a result of collaboration between the customer, businessperson and the developers.

The Planning Game – Advantages

The Planning Game has the following advantages-

- Reduction in time, wasted on useless features
- Greater customer appreciation of the cost of a feature
- Less guesswork in planning

Short Releases

You should put a simple system into production quickly, and then release new versions in very short cycles. Every release should be as small as possible, so that it is-

- Achievable in a short cycle
- Contains the most valuable and immediate business requirements
- A working system

The duration of the short cycle may vary with the software that needs to be built. However, it needs to be ensured that the minimum possible duration is chosen.

Short Releases – Advantages

The advantages of Short Releases are-

- Frequent feedback
- Tracking
- Reduce chance of overall project slippage

Metaphor

According to Cambridge online dictionary- A Metaphor is an expression, often found in literature that describes a person or object by referring to something that is considered to have similar characteristics to that person or object. For example, 'The mind is an ocean' and 'the city is a jungle' are both Metaphors.

You should guide the entire development with a simple shared story of how the whole system works. You can think of metaphor as the architecture of the system to be built in a way that it is easily understandable by everyone involved in the development.

The metaphor consists of domain specific elements and shows their interconnectivity. The language used is the domain language. To identify the technical entities, the words used in the metaphor need to be taken consistently.

As the development proceeds and the metaphor matures, the whole team will find new inspiration from examining the metaphor.

The goal of a good architecture is to give everyone a coherent story within which to work, a story that can easily be shared by both the business and the technical members. Hence, in Extreme Programming, by asking for a metaphor, we are likely to get an architecture that is easy to communicate and elaborate.

Metaphor – Advantages

The advantages of Metaphor are-

- Encourages a common set of terms for the system
- Reduction of buzz words and jargon
- A quick and easy way to explain the system

Simple Design

The system should be designed as simply as possible at any given moment. Extra complexity is removed as soon as it is discovered.

The right design for the software at any given time is the one that-

- Runs all the tests
- Has no duplicated logic like parallel class hierarchies
- States every intention important to the developers
- Has the fewest possible classes and methods

To get a simple design, eliminate any design element that you can, without violating the first three rules. This is opposite to the advice- Implement for today, design for tomorrow. If you believe that the future is uncertain and you can quickly enhance the design, then do not put any functionality on speculation.

Simple Design – Advantages

The advantages of Simple Design are-

- Time is not wasted adding superfluous functionality
- Easier to understand what is going on
- Refactoring and collective ownership is made possible
- Helps keep the programmers on track

Testing

The developers continually write unit tests, which need to pass for the development to continue. The customers write tests to verify that the features are implemented. The tests are automated so that they become a part of the system and can be continuously run to ensure the working of the system. The result is a system that is capable of accepting change.

Testing – Advantages

The advantages of testing are-

- Unit testing promotes testing completeness
- Test-first gives developers a goal
- Automation gives a suite of regression tests

Refactoring

When implementing a feature, the developers always ask if there is a way of changing the existing code to make adding the feature simple. After they have added a feature, the developers ask if they now can see how to make the code simpler, while still running all of the tests. They restructure the system without changing its behavior to remove duplication, improve communication, simplify, or add flexibility. This is called Refactoring.

Refactoring – Advantages

The advantages of Refactoring are-

- Prompts the developers to proactively improve the product as a whole
- Increases the developer knowledge of the system

Pair Programming

In Pair programing, the entire code is written with two developers at one machine, with one keyboard and one mouse.

There are two roles in each pair-

- The first developer (the one with the keyboard and the mouse) thinks about the best way to implement this method right here.
- The other developer is thinks more strategically-
    - Is this whole approach going to work?
    - What are some other test cases that might not work yet?
    - Is there some way to simplify the whole system so the current problem just disappears?

The pairing is dynamic. It means that the two Roles A and B may exchange their places, or they may pair up with other team members. More often, anyone on the team will do as a partner. For example, if you have a responsibility for a task in an area that is unfamiliar to you, you might ask someone with recent experience to pair with you.

Pair Programming – Advantages

The advantages of Pair Programming are-

- Two heads are better than one
- Focus
- Two people are more likely to answer the following questions-
  - Is this whole approach going to work?
  - What are some test cases that may not work yet?
  - Is there a way to simplify this?

Collective Ownership

In Extreme Programming, the entire team takes responsibility for the whole of the system. Not everyone knows every part equally well, although everyone knows something about every part.

If a pair is working and they see an opportunity to improve the code, they go ahead and improve it.

Collective Ownership – Advantages

The advantages of Collective ownership are-

- Helps mitigate the loss of a team member who is leaving.
- Promotes the developers to take responsibility for the system as a whole rather than parts of the system.

Continuous Integration

Code is integrated and tested many times a day, one set of changes at a time. A simple way to do this is to have a machine dedicated to integration. A pair with code ready to integrate-

- Sits when the machine is free.
- Loads the current release.
- Loads their changes (checking for and resolving any collisions).
- Runs the tests until they pass (100% correct).

Integrating one set of changes at a time helps in knowing who should fix a test that fails. The answer- is the present pair, since the last pair left the tests at 100%. They may have to throw away what they have done and start all over, as they might not have known enough to code that feature.

Continuous Integration – Advantages

The advantages of Continuous Integration are-

- Reduces the duration, which is otherwise lengthy.
- Enables the short releases practice as the time required before release is minimal.

40-Hour Week

Extreme Programming emphasizes on the limited number of hours of work per week for every team members, based on their sustainability, to a maximum of 45 hours a week. If someone works for more time than that, it is considered as overtime. Overtime is allowed for at most one week. This practice is to ensure that every team member be fresh, creative, careful and confident.

40-Hour Week – Advantages

The advantages of 40-hour week are-

- Most developers lose effectiveness past 40 hours.
- Value is placed on the developers' well-being.
- Management is forced to find real solutions.

On-Site Customer

Include a real, live user on the team, available full-time to answer the questions, resolve disputes and set small-scale priorities. This user may not have to spend 40 hours on this role only and can focus on other work too.

On-Site Customer – Advantages

The advantages of having an onsite customer are-

- Can give quick and knowledgeable answers to the real development questions.
- Makes sure that what is developed is what is needed.
- Functionality is prioritized correctly.

Coding Standards

Developers write all code in accordance with the rules emphasizing-

- Communication through the code.
- The least amount of work possible.
- Consistent with the "once and only once" rule (no duplicate code).
- Voluntary adoption by the whole team.

These rules are necessary in Extreme Programming because all the developers-

- Can change from one part of the system to another part of the system.
- Swap partners a couple of times a day.
- Refactor each other's code constantly.

If the rules are not followed, the developers will tend to have different sets of coding practices, the code becomes inconsistent over time and it becomes impossible to say who on the team wrote what code.

Coding Standards – Advantages

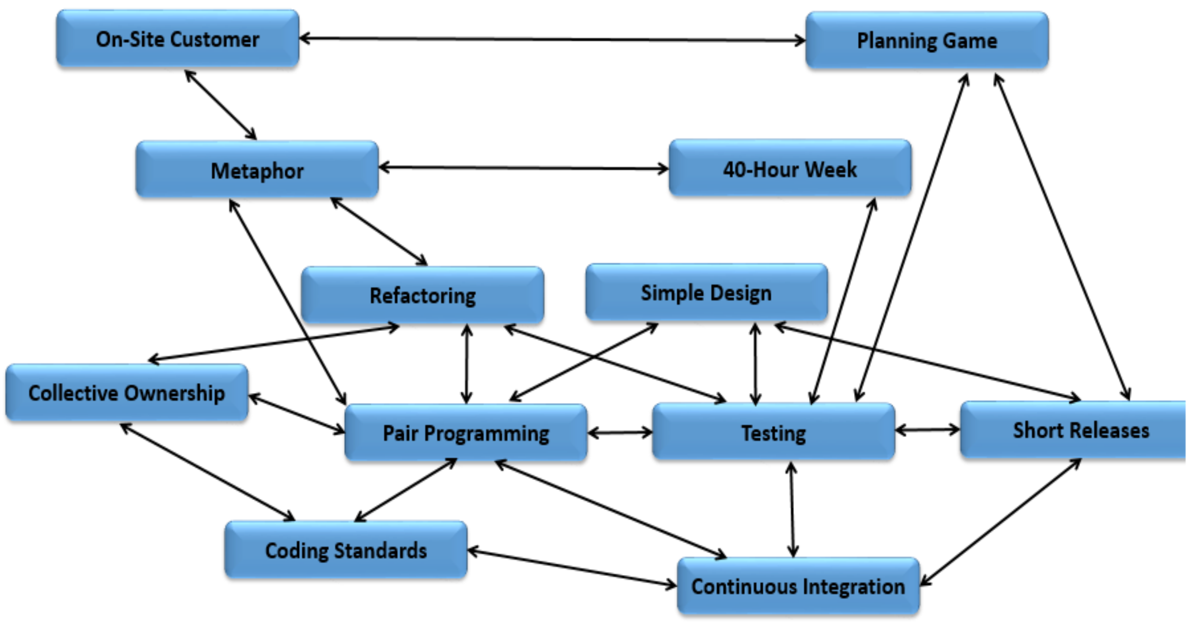
The advantages of Coding Standards are-

- Reduces the amount of time developers spend reformatting other peoples' code.
- Reduces the need for internal commenting.
- Calls for clear, unambiguous code.

4. XP – Supporting Practices

The Extreme Programming practices if implemented in isolation can be weak and thus, can fail. In Extreme Programming, all the practices need to be considered as a whole, so that they support each other. The weakness of one is covered by the strengths of others.

In this chapter, we will focus on the possible weakness of each practice if implemented in isolation. We will see how Extreme Programming can support this practice to overcome the weakness when implemented in conjunction with other practices.



The Planning Game – Support from other XP Practices

In this section, we will see the weaknesses of the Planning Game are and how the other XP practices support it.

The Planning Game – Disadvantages

The disadvantages of the Planning Game are that you cannot possibly start development with only a rough plan and you cannot constantly update the plan as it would take too long and upset the customers.

The Planning Game with other XP practices

The other XP practices support the planning game in the following way-

- In planning game, the customer is also involved in updating the plan, based on the estimates provided by the developers.

- You make short releases so that any mistake in the plan would have at the most a few weeks or months of impact.
- Your customer sits with the team, so they could spot potential changes and opportunities for improvement quickly (on-line customer).
- Continuous testing helps the developers and the customer to decide what is required immediately.

Thus, you can start development with a simple plan, and continually refine it as you go along.

Short Releases – Support from other XP Practices

Short Releases – Disadvantages

You cannot possibly go into production after a few months. You certainly cannot make new releases of the system on cycles ranging from daily to every couple of months. This is because you take time to absorb new requirements, changes into the current code.

Short Releases with other XP practices.

The other XP practices support Short Releases in the following way-

- The planning game helps you work on the most valuable stories, so even a small system will have business value.
- Continuous integration makes the cost of packaging a release minimal.
- Testing reduces the defect rate enough so that a lengthy test cycle is not required before a release.
- You can make a simple design, sufficient for this release, not for all time.

Thus, you can make Short Releases, soon after the development begins.

Metaphor – Support from other XP Practices

You cannot possibly start development with just a metaphor. It may not have enough details there, and you may be wrong.

Metaphor with other XP practices

The other XP practices support Metaphor in the following way-

- With pair programming, you will have quick concrete feedback from the implemented code and tests about whether the metaphor is working fine.
- Your on-site customer is comfortable talking about the system in terms of the metaphor.

- Continuous refactoring allows you to refine your understanding of what the metaphor means in implementation.
- Simple design helps you to have a mapping with the metaphor.

Thus, you can start development with just a metaphor.

Simple Design – Support from other XP Practices

Simple Design - Disadvantages

You cannot possibly have just enough design for today's code and your design may not continue to evolve the system.

Simple Design with other XP practices.

The other XP practices support Simple Design in the following way-

- Refactoring allows you to make changes.
- With an overall metaphor, you are sure that the future design changes would tend to follow a convergent path.
- Pair programming helps you to be confident that you are making a simple design that works.
- 40-hour week helps you to be focused on the right design.
- Continuous unit testing and customer testing ensures that your simple design is on the track.

Thus, you can do the best possible design for today, without speculation.

Testing – Support from other XP Practices

Testing - Disadvantages

You may think that-

- You cannot possibly write all those tests.
- It can take too much time to write the tests.
- Developers will not write the tests.

Testing with other XP practices

The other XP practices support Testing in the following way-

- Simple design makes writing tests easy.
- Refactoring allows you to decide on what tests are necessary.

- With pair programming, even if you cannot think of another test, your partner can. You can allow your partner to run the tests handing over the keyboard and you feel confident when you see the tests all running.

- Collective ownership ensures that a developer with required skills is working on any complex part for coding and testing.

- Continuous integration and immediately running the tests for every set of changes made by a pair ensures-

    - That the new code works if 100% tests pass, or

    - That if any test fails, it is that pair's code that is failing the system so that the changes can be immediately reversed and the pair can start afresh with coding with the clarity on the feature they are implementing.

- Short releases ensure that a working system is available for the customer to run the tests and give feedback.

- On-line customer will have time to run all the tests and provide feedback immediately on the working system.

- Planning game ensures to take the feedback from the customer after testing to plan for the next release.

Thus, the developers and the customers will write tests. Further, the tests are automated to ensure the working of the rest of the Extreme Programming.

Refactoring – Support from other XP Practices

Refactoring - Disadvantages

You cannot possibly **refactor** the design of the system all the time. It would-

- Take too long.
- Be too hard to control, and
- Most likely break the system.

Refactoring with other XP practices

The other XP practices support Refactoring in the following way-

- With collective ownership, you can make changes wherever they are needed.

- With coding standards, you need not reformat before refactoring.

- With pair programming, you can have the courage to tackle a tough refactoring.

- With a simple design, the refactoring is easier.

- With metaphor, you can communicate easily.
- With testing, you are less likely to break something without knowing it.
- With continuous integration, if you accidentally break something, or if your refactoring conflicts with someone else's work, you come to know in a few hours.
- With 40-hour week, you are rested and so you have more courage and are less likely to make mistakes.

Thus, you can **refactor** whenever you see the chance to –

- Make the system simpler
- Reduce duplication
- Communicate more clearly

Pair Programming – Support from other XP Practices

Pair Programming - Weakness

You cannot possibly write all the code in pairs. It will be too slow. It makes the situation difficult if two people do not get along well.

Pair Programming with other XP practices.

The other XP practices support pair programming in the following way-

- The coding standards reduce conflicts.
- With 40-hour work, everyone is fresh and focused, reducing further the chance of unnecessary discussions.
- The pairs write tests together, giving them a chance to align their understanding before tackling the implementation part.
- Metaphor helps the pairs to ground their decisions about naming and the basic Design
- Simple Design enables the pair to have a common understanding.
- Refactoring helps the pair to discuss and make combined decisions in making the system simple.
- Continuous Integration gives a chance to the pair to correct in case of any mistakes and hence a partner will not object when the other does some experimentation.
- Collective Ownership enables the team to mix and match and allows them to maintain a cordial relationship.

Thus, you can write all the code in pairs. On the other hand, if the team works in solo, they are more likely to make mistakes, overdesign and overlook the other practices.

Collective Ownership – Support from other XP Practices

Collective Ownership – Disadvantages

You cannot possibly have everybody changing anything anywhere in the System. For, it is possible to break the System unknowingly and the cost of integration will go up dramatically.

Collective Ownership with other XP practices

The other XP practices support Collective Ownership in the following way-

- Continuous Integration reduces the chances of conflicts.
- Testing reduces the chance of breaking things accidentally.
- With Pair Programming, you are less likely to break the code, and developers learn faster what they can profitably change.
- With Coding Standards, you will not have conflicts on the code.
- With Refactoring, you maintain the system simple so that everyone understands it.

Thus, you can have anyone change code anywhere in the system when they see the chance to improve it. On the other hand, without collective ownership the rate of evolution of the design slows dramatically.

Continuous Integration - Support from other XP Practices

Continuous Integration – Disadvantages

You cannot possibly integrate after only a few hours of work, as integration takes long and there are too many conflicts and chances to accidentally break something.

Continuous Integration with other XP practices

The other XP practices support Continuous Integration in the following way-

- Testing quickly helps you to know that you have not broken anything.
- With pair programming, there are half as many streams of changes to integrate.
- With refactoring, there are smaller pieces, reducing the chance of conflicts.
- With coding standards, the code will be consistent.
- Short releases ensure immediate feedback on the system.

- Collective ownership ensures that whoever changes the code and integrates will have a whole view of the system.

Thus, you can integrate after a few hours. On the other hand, if you do not integrate quickly, then the chance of conflicts rises and the cost of integration goes up steeply.

40-Hour Week – Support from other XP Practices

40-Hour Week – Disadvantages

You cannot possibly work 40-hour weeks. You cannot create enough business value in 40 hours.

40-Hour Week with other XP practices

The other XP practices support the 40-hr week in the following way-

- The planning game provides you what more valuable work to do.

- The combination of the planning game and testing ensures that you have to work on only what you thought.

- Simple design and refactoring allows you to stay focused and complete on time.

- Pair programming helps you to work on what you can do, sharing the other work with your partner.

- The practices as a whole help you develop at top speed, and hence you cannot go any faster.

Thus, you can produce enough business value in 40-hour weeks. On the other hand, if the team does not stay fresh and energetic, then they won't be able to execute the rest of the practices.

On-Site Customer – Support from other XP Practices

On-Site Customer – Disadvantages

You cannot possibly have a real customer full-time on the team, not producing any value. They can produce far more value for the business elsewhere.

On-Site Customer with other XP practices

The other XP practices support the on-site customer in the following way.

They can produce value for the project-

- In Planning Game, by making priority and scope decisions for the developers.

- With Metaphor, to bring clarity for the developers on the domain.
- In testing, by writing acceptance tests and by performing acceptance testing after every short release.

Thus, they can produce more value for the organization by contributing to the project.

Coding Standards – Support from other XP Practices

Coding Standards – Disadvantages

You cannot possibly ask the team to code to a common standard as developers are usually individualistic.

Coding Standards with other XP practices

The other XP practices support Coding Standards in the following way-

- Pair Programming makes them adapt to the necessary Coding Standards easily.
- Continuous Integration enforces them to follow standard so that the code is consistent.
- Collective Ownership encourages them to align to the standard to make changes when and wherever necessary.

5. XP – Evolving Practices

Extreme Programming has been evolving since its inception and Extreme Programming practices are being found to be effective in the other agile methodologies also.

Following table shows how the Extreme Programming practices are being evolved.

Extreme Programming Practice	Evolution
On-Site Customer	Whole Team
The Planning Game	Release Planning Iteration Planning
Testing	Acceptance Testing Unit Testing Test-Driven Development
Refactoring	Design Improvement
40-Hour Week	Sustainable Pace

On-Site Customer – Whole Team

Extreme Programming relies on a project community with emphasis on team-centric approach. All the contributors to an Extreme Programming project, including the customer are one team.

The customer provides requirements, sets priorities and steers the project. This enables the customer to understand the practical details of development and set the priorities and expectations accordingly. This changes from “As customer asks the development” to “As the customer understands and cooperates with the development”.

The project goals are a shared responsibility and the development is an ongoing conversation across the whole team. It is a cooperative game of invention and communication. It is found that face-to-face communication is the most efficient and the most effective method in the path of development and eliminates wait times and delays.

Planning Game – Release and Iteration Planning

Incremental project planning is found to be effective since it promotes accurate plans. More and better information is learned as the development progresses based on the actual performance. Develop a rough plan first and refine it incrementally.

Release planning sets long-term goals, with an overall big-picture at hand. The customer presents required features, the developers estimate and the release dates are mutually agreed and committed. As the release plan is revised after every release, it becomes more precise as the project progresses.

Iteration planning sets short-term time-boxes with iterations, typically ranging from 1 week to 1 month. The main objective of iteration planning is a working software at the end of each iteration. The developers choose the features/stories for the iteration, break them down into tasks, estimate tasks and commit to the allocated tasks. Sustainable pace is ensured by balancing the load factor across the team, considering 40-hour week.

Acceptance Testing

The customer writes one or more automated acceptance tests for a feature, to ensure that the system implements the desired features correctly. The acceptance tests are written along with the stories and provided prior to implementation.

Team automates these tests to verify that a feature is implemented correctly. Once the test runs, the team ensures that it keeps running correctly thereafter, at the time of regression, by executing all the acceptance tests implemented till then.

Acceptance tests provide non-ambiguous specifications of functional requirements. Further, the percentage of acceptance tests passing measures the release completion, with no last-minute surprises.

The system always improves and never backslides.

Unit Testing

The developer writes unit tests with sufficient coverage, incorporating the intent and usage of the code modules and methods. Unit tests are automated, with clear pass/fail. Most languages have a xUnit framework (e.g., nUnit, jUnit).

All unit tests are executed very frequently and the code cannot be checked-in until all unit tests pass. The unit test results also help in refactoring.

Test Driven Development

Test Driven Development is being considered as the most innovative Extreme Programming practice.

In Test Driven Development, the developer writes a unit test prior to writing code. The intent is to make the unit test fail. As the code is not yet implemented, the unit test fails. The developer writes just enough code to make the unit test pass and then, the developer refactors to ensure that the code is simple and clean (with no duplications and complexity).

The developer iterates till the coding is complete and the acceptance test passes.

The Unit Tests are all collected together and each time a pair integrates and releases code to the repository, the pair needs to ensure that every test runs correctly. If any test fails, the pair knows that it is their code that needs to be corrected as the prior integrations passed without any defects.

Test Driven Development tends to result in 100% unit test coverage and ensures the code to be simple and minimal.

Refactoring – Design Improvement

Refactoring allows the design to evolve incrementally keeping it simple, removing the duplications and complexity as you notice. It improves the design of the existing code without changing its functionality by refactoring.

Refactoring should be driven by learning from new implementations. It is advised to do the Refactoring just after writing new code. Refactoring drives the code towards higher-level design patterns and is supported by testing.

40-Hour Week – Sustainable Pace

Work at a pace that can be sustained indefinitely. Sustainable Pace ensures the human contribution to the success of the project, considering the facts that-

- Fatigue and stress reduce productivity and also the quality of the product. It may lead to staff-turnover.
- Development does not stop with a sprint, and it should be aiming at long term objectives
- Unless the team agrees on expectations there will not be commitment and accountability.
- Exact hours are not as important as the ability to perform.
- Micro-management should be avoided, while ensuring availability as and when required

6. XP – Process Cycle

Extreme Programming is an Agile process.

Extreme Programming is an Agile process because it -

- Emphasizes plenty of communication and feedback-
  - Within the team (pair programming, collective code ownership, simple design)
  - With the customer (on-site customer and acceptance testing)
  - For release planning (with customer and developers participating in estimation)

- Extreme Programming employs a coach whose job is noticing when people are not communicating and reintroduce them.

- Embraces change with-
  - Frequent iterations (short releases)
  - Designing and redesigning easily (simple design)
  - Coding and testing continuously (pair programming)
  - Keeping the customer constantly involved (on-line customer)

- Delivers working product to the customer in short iterations (short releases).

- Eliminates defects early, thus reducing costs (pair programming)
  - Code reviews
  - Unit testing
  - Integration for every set of changes and testing

Extreme Programming Process Cycle

Extreme Programming is iterative and incremental and is driven by Time-Boxed Cycles. Therefore, the rhythm of the Extreme Programming process is crucial.

Extreme Programming has the following activity levels-

- Product Life Cycles
- Releases
- Iterations
- Tasks
- Development
- Feedback

Each of the activity levels provides the minimal inputs required for the next level. They are-

- Product life cycle activities provide inputs for release cycles.
- Release planning sessions provide inputs for iteration cycles.
- Iteration planning sessions provide inputs for task cycles.
- Task development provides inputs for development episodes.
- Development produces the product.

Feedback is a continuous activity throughout the project and across all the above activity levels.

Product Life Cycles

This is also referred to as the Exploration phase. It involves feature set definition and planning. The customer arrives at high value requirements and the requirements are given as user stories.

Stories are the primary deliverable of this level activity.

Releases

This is also referred to as the Commitment phase. In this activity-

- The whole team gathers so that-
  - Progress is reviewed.
  - New requirements can be added and/or existing requirements can be changed or removed.
  - Customer presents stories.
  - Stories are discussed.
- The developers determine the technical approach and risk. They provide first-level estimates and options.
- The customer prioritizes the stories and chooses target release time box.
- The customer and developers commit themselves to the functionality that are to be included and the date of the next release.
- The developers-
  - Arrange stories into probable iterations.
  - Include defect fixes from acceptance testing of the previous release.
  - Begin the iterations.

Release plan is the primary deliverable of this level activity.

Iterations

This is also referred to as the Steering phase. The whole team gathers so that the progress is reviewed and the plan can be adjusted. The customer presents stories for the iteration and the stories are discussed in greater detail.

The iteration Plan is the primary deliverable of this activity.

The developers-

- Determine detailed technical approach.
- Create task list for each story.
- Begin the development.
- Deploy the system as far as possible each iteration.

Deployable System is the final deliverable of this activity.

Tasks

The developers Sign-up for the tasks and begin development episodes to implement the stories. They ensure the tasks for the iteration are complete. The developers also ensure that the stories for the iteration are complete with acceptance tests.

Development

The developers form pairs, which can be a continuous and dynamic activity.

Each Pair-

- Verifies their understanding of the story.
- Determines detailed implementation approach, ensuring simple design.
- Begins test-driven development, i.e., write unit test, implement code to pass the unit test, refactor to make the code simple.
- Integrates their code to the system code base at appropriate intervals.
- Reviews the progress frequently.

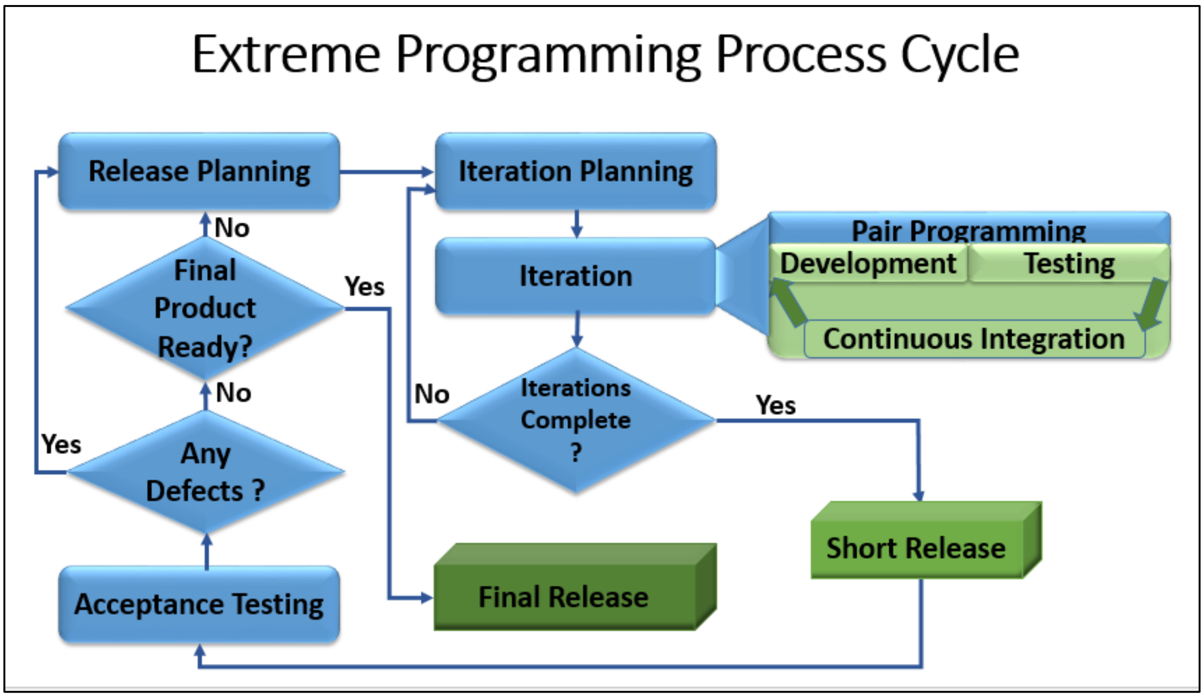
Feedback

Pairs constantly communicate within themselves and outward to the team as well. On-line customer is also involved in the communication on a continuous basis. Certain teams resort to daily stand-up meetings to discuss the overall team status quickly and the possible re-synchronization and micro-planning if necessary.

The iteration and release reviews provide overall status and points for process adjustment and improvement.

- Development episodes may cause rethinking of tasks.
- Task development may cause rethinking of stories.
- Story re-estimation may cause iteration changes or recovery.
- Iteration results may cause changes to release plan.

The Extreme Programming process cycle is illustrated below.



7. Pair Programming

Pair programming is a style of programming in which two programmers work side-by-side at one computer, sharing one screen, keyboard and mouse, continuously collaborating on the same design, algorithm, code or test.

One programmer, termed as the **driver**, has control of the keyboard/mouse and actively implements the code or writes a test. The other programmer, termed as the **navigator**, continuously observes the work of the driver to identify defects and also thinks strategically about the direction of the work.

When necessary, the two programmers brainstorm on any challenging problem. The two programmers periodically switch roles and work together as equals to develop a software.

Pair Programming – Advantages

The significant advantages of Pair Programming are-

- Many mistakes are detected at the time they are typed, rather than in QA Testing or in the field.
- The end defect content is statistically lower.
- The designs are better and code length shorter.
- The team solves problems faster.
- People learn significantly more about the system and about software development.
- The project ends up with multiple people understanding each piece of the system.
- People learn to work together and talk more often together, giving better information flow and team dynamics.
- People enjoy their work more.

Pair Programming Experiments

Use of pair programming practice has been demonstrated to improve the productivity and quality of software products.

Pair Programming research reveals that-

- Pairs use no more man-hours than singles.
- Pairs create fewer defects.
- Pairs create fewer lines of code.
- Pairs enjoy their work more.

University of Utah conducted experiments on pair programming. The results revealed that-

- Pairs spent 15% more time on the program than individuals.
- Code written by pairs consistently passed more test cases than code written by individuals.
- Pairs consistently implemented the same functionality produced by individuals in fewer lines of code**.**
- Learning how to program in an environment where there are rapidly tangible results is fun and allows one to learn faster**.**

Adapting to Pair Programming

Most programmers are used to solitary work and often resist the transition to pair programming. However, with practice they can ultimately make this transition.

According to Laurie A. Williams and Robert R. Kessler, in their book, 'All I Really Need to Know about Pair Programming I Learned in Kindergarten', it is well explained of how to nurture the skills that we all have learnt in Kindergarten to establish team cohesion, in general and pair programming in particular.

The transition and on-going success as a pair programmer often involves practicing everyday civility.

The following sections are an excerpt of this publication that help you in becoming effective pair programmers.

Kindergarten lessons

In Kindergarten, we have learnt the following-

- Share everything
- Play fair
- Do not hit people
- Put things back where you found them
- Clean up your own mess
- Do not take things that are not yours
- Say you are sorry when you hurt somebody
- Wash your hands before you eat
- Flush
- Warm cookies and cold milk are good for you
- Live a balanced life – learn some and think some and draw and paint and sing and dance and play and work every day some
- Take a nap every afternoon

- When you go out into the world, watch out for traffic, hold hands and stick together
- Be aware of wonder

Next, we look at the principles of Pair Programming in the context of the above given teachings.

Share everything

In pair programming,

- Two Programmers sit together and jointly produce one artifact (design, algorithm, code, etc.)

- One person is typing or writing, the other is continually reviewing the work. Both
    - Are equal participants in the process
    - Responsible for every aspect of the artifact
    - Own everything

Play fair

In pair programming,

- One person drives, i.e. has control of the keyboard or is recording design ideas, while the other is continuously reviewing the work.

- They switch these roles periodically, even when one of them is significantly more experienced than the other, to ensure equal participation.

- While the person who is driving is thinking about implementation, the other continuously reviews code, thinks about a possible simpler design that is possible, how the current development fits in the overall system as of date.

Do not hit your partner

In pair programming,

- Ensure that your partner stays focused and on-task.

- You stay focused and on-task.

- Ensure your partner follows the prescribed coding standards and thus maintains the commitment to the rest of the team.

In the pair programming survey, it is found that tremendous productivity gains and quality improvements are realized. This is because-

- Each one keeps their partner focused and on-task with no possibility of slack off.
- Each artifact is reviewed continuously as it is being produced ensuring quality.

Put things back where they belong

In Pair Programming,

- You need to believe in your skills and your partner's skills as well. Any negative thoughts in this aspect are to be put in trash can.
- You have to be sure that you express what you know and are open to learn from your partner when required. You can learn from your partner by observing him or taking his feedback instantly.
- You need to have the confidence that-
    - Wherever there is a possibility of lagging, you can immediately pick up from your partner.
    - Together as a pair, you can solve problems that you could not solve alone.
    - You can help improve each other's skills.

Clean up your mess

In Pair Programming, with the 'watch over the shoulder' technique,

- You will find that it is amazing to know how many obvious but unnoticed defects are noticed by your partner.
- You can remove these defects without the natural animosity that might develop in a formal inspection meeting.
- Characterizing defect prevention and defect removal efficiency.

Do not take things too seriously

Having a partner to review design and coding continuously and objectively is a very beneficial aspect of pair programming. In pair programming, you need to ensure that you work without excess ego or too little ego.

This is necessary because,

- Excess ego can manifest itself in two ways-
    - Having a "my way or the highway" attitude can prevent the programmer from considering other's ideas.
    - Being defensive can cause a programmer not to receive constructive criticism or to view this criticism as mistrust.

Both these ways of ego manifestation damage the collaborative relationship.

- On the other hand, a person who always agrees with the partner so as not to create tension also minimizes the benefits of collaborative work. For favorable idea exchange, there should be some healthy disagreement/debate when required.

Thus, a fine balance between displaying too much and too little ego is necessary. Effective pair programmers groom this balance during an initial adjustment period that can take hours or days, depending on the individuals, the nature of work and their past experience with pair programming.

Say you are sorry when you hurt somebody while moving furniture

The programmers must be able to sit side-by-side and program, simultaneously viewing the computer screen and sharing the keyboard and the mouse. Extreme programmers have a "slide the keyboard/don't move the chairs" rule.

To ensure effective communication, both within a collaborative pair and with other collaborative pairs, without much effort, programmers need to see each other, ask each other questions and make decisions on things such as integration issues. Programmers also benefit from overhearing other conversations to which they can have vital contributions.

Disown skepticism before you start

For success of pair programming, it is necessary that both the partners understand the value of collaboration in programming, the benefits, and the joy of the experience. Any skepticism in this regard needs to be stopped in the beginning itself.

Experience has shown that having one programmer, very positive and/or experienced in pair programming, can lead the pair to become one jelled collaborative team victoriously.

- The production of such a team is greater than that of the same people working in un-jelled form.
- The enjoyment that people derive from their work is greater than what you would expect, given the nature of the work itself.
- Once a team begins to jell, the probability of success goes up dramatically.

Flush

The pair programmers can work on something independently. However, when they rejoin, they have to either review the independent work before incorporating it or flush and rewrite the independent work along with continuous review of the work, which identifies additional defects.

Never incorporate any independent work without the review by the partner. This is for the reason that studies have indicated that the independent work has defects as compared to the work produced by the pair.

Warm cookies and cold milk are good for you

Pair programmers keep each other continuously focused and on-task. It can be very intense and mentally exhausting. Hence, periodically take a break to maintain the stamina for another round of productive Pair Programming.

During the break, it is best to disconnect from the task and approach it with a freshness when restarting. Suggested activities are checking email, making a phone call, browsing the web, or taking a Snack-break.

Live a balanced life

Communicating with others on a regular basis is the key for leading a balanced life. Informal discussions with your partner and with other programmers allows exchange of effective ideas and efficient transfer of information.

Take a break from working together every afternoon

It is not necessary to work separately every afternoon, but it is acceptable to work alone 10-50% of the time. This is because-

- Many programmers prefer to do experimental prototyping, tough, deep-concentration problems and logical thinking alone.
- Simple, well-defined and routine coding is done more efficiently by a solitary programmer and then reviewed with a partner.

Watch out for traffic, hold hands and stick together

In Pair Programming,

- There should be no competition between the two. Both must work together as if the artifact is produced by a single mind.

- The Partners need to trust each other's judgement and each other's loyalty to the team.

- A partner should never blame the other partner for any problems or defects.

Be aware of the power of two brains

When two are working together, each has their own set of knowledge and skills, comprising of-

- A common set of this knowledge and these skills that enables them to communicate effectively.

- Unique skills that allow them to contribute to accomplish their tasks.

Together, a pair will-

- Come up with more than twice as many possible solutions than the two would have when working alone.

- Proceed more quickly to narrow in on the best solution.

- Implement it more quickly and with better quality.

Thus, pair programming is a powerful technique as there are two brains concentrating on the same problem all the time. It forces one to concentrate fully on the problem at hand.

8. XP – Roles

In Extreme Programming, the emphasis is on the collaboration of the whole team, collocated and is in continuous communication.

However, certain roles are required for an extreme programming project to work and the persons who take these roles take the corresponding responsibilities and are accountable for their contribution to these roles. It is advised to allocate the right people for the roles rather than trying to change the people to fit into those roles.

Roles in Extreme Programming

The Roles that have been found effective in Extreme Programming are-

- Developer (also called Programmer by some teams)
- Customer
- Manager (also called tracker)
- Coach

Developer

The role of developer is the most important one in Extreme Programming. To be a developer in Extreme Programming, you need to be accustomed with the following-

- **Developer Rights**
  - You have the right to know what is needed, with clear declarations of priority.
  - You have the right to produce quality work at all times.
  - You have the right to ask for and receive help from peers, superiors, and customers.
  - You have the right to make and update your own estimates.
  - You have the right to accept your responsibilities instead of having them assigned to you.
- Major responsibilities that you will be accountable for-
  - Estimate stories
  - Define tasks from stories
  - Estimate tasks
  - Write unit tests
  - Write code to pass the written unit tests

    - Perform unit testing
    - Refactor
    - Integrate continuously

- **Developer Skills**
    - Pair Programming
    - Communication – that is necessary and to the sufficient detail
    - Always use the metaphor to use the right names
    - Code only what is required.
    - Maintain simplicity

- **Collective Ownership**
    - If someone changes the code that you wrote, in whatever part of the system, you have to trust the changes and learn. In case the changes are wrong-headed, you are responsible for making things better, but be careful not to blame anyone.
    - Be prepared to acknowledge your fears. Remember that you are part of a team and courage is required for the success of Extreme Programming.

- Wear different hats as a developer in the team such as-
    - Programmer.
    - Architect and designer.
    - Interface architect/UI designer.
    - Database designer and DBA.
    - Operations and network designer.

- Sometimes one of the developers has to wear the hat of a tester.
    - Help the customer choose and write functional tests.
    - Run the functional tests regularly.
    - Report the test results.
    - Ensure the testing tools run well.

Customer

In Extreme Programming, the customer role is as crucial as the developer role as it is the customer who should know what to program, while the developer should know how to program.

This triggers the necessity of certain skills for the customer role-

- Writing required stories to the necessary and sufficient detail.
- Developing an attitude for the success of the project.
- Influencing a project without being able to control it.
- Making decisions that are required time to time on the extent of functionality that is required.
- Willing to change decisions as the product evolves.
- Writing functional tests.

In Extreme Programming, the customer is required to be in constant communication with the team and speak as a single voice to the team. This is required since-

- Customer could be multiple stakeholders.
- Customer could be a community.
- Customer is not always the PRINCIPAL (proxies).
- Customer can be a team with the following potential members-
  - Product Managers
  - Marketing, Sales
  - Business Analysts
  - End Users, their Managers
  - Business/System Operations

The Major Responsibilities of the Customer are-

- Write user stories
- Write functional tests
- Set priorities on the stories
- Explain stories
- Decide questions about the stories

The customer is accountable for these responsibilities.

Manager

In Extreme Programming, the major responsibilities of the manager are-

- Define the rules of planning game.
- Familiarize the team and the customer on the rules of the planning game.
- Monitor the planning game, fix any deviations, modify the rules if and when required.

- Schedule and conduct release planning and iteration planning meetings.
- Participate with the team while they estimate in order to provide feedback on how reality conformed to their prior estimates, both at team level and individual level that eventually help them to come up with better estimates next time.
- Ensure that the team is working towards the next release as they progress through the iterations with committed schedule and committed functionality.
- Track functional testing defects.
- Track the actual time spent by each team member.
- Adapt an ability to get all the required information without disturbing the team's work. The Manager is Accountable for these Responsibilities.

Coach

Extreme Programming is the responsibility of everyone in the team. However, if the team is new to Extreme Programming, the role of a coach is crucial.

The responsibilities of the coach are-

- Understand, in depth, the application of Extreme Programming to the project.
- Identify the Extreme Programming practices that help in case of any problem.
- Remain calm even when everyone else is panicking.
- Observe the team silently and intervene only when a significant problem is foreseen and make the team also see the problem.
- See to that the team is self-reliant.
- Be ready to help.

9. XP – Activities and Artifacts

In this chapter, we will understand the Extreme Programming activities and artifacts.

XP – Activities

In Extreme Programming, the four basic activities are-

- Coding
- Testing
- Listening
- Designing

Coding

In pair programming, coding is considered the heart of development. You code because if you do not code, at the end of the day you have not done anything.

Testing

In pair programming, testing needs to be done to ensure that the coding is done. If you do not test, you do not know when you have finished coding.

Listening

In pair programming, you listen to know what to code or what to test. If you do not listen, you do not know what to code or what to test.

Designing

In pair programming, you design so that you can keep coding and testing and listening indefinitely, as good design allows extension of the system, with changes in only one place.

These activities are performed during-

- Release Planning
- Iteration Planning
- Implementation

Release Planning

In Release planning, both the customer and the developers jointly arrive at the plan for the next release, agreeing on the user stories for the release and the next release date.

Release Planning has three phases-

- Exploration Phase
- Commitment Phase
- Steering Phase

Release Planning – Exploration Phase

In Exploration phase,

- The customer provides a shortlist of high-value requirements for the system.
- The developers gather requirements and estimates the work impact of each of those requirements.

The Requirements are written down on user story cards. For writing a story, the customer will come with a problem in a meeting with the developers. Developers will try to define this problem and get requirements. Based on this discussion, a story will be written by customer, pointing out what they want a part of the system to do. It is important that the developers have no influence on this story.

Active Listening is significant in this phase, as

- The Developers need to understand "What the customer is asking for" and "What requirements are of high value"
- The customer needs to understand along with the developers about what scenarios contribute to these values to write the story.

Though the customer writes the requirements on user story cards, you need to listen to

- Obtain clarity
- Avoid ambiguity
- Express yourself if there are possible gaps in understanding

This is possible only with verbal communication.

Release Planning – Commitment Phase

In Commitment phase, the customer and developers will commit themselves to the functionality that will be included and the date of the next release.

This phase involves the determination of costs, benefits, and schedule impact. In this phase,

- The customer sorts the user stories by business value.
- The developers sort the stories by risk.
- Developers determine the effort and duration (estimates) required for implementation of the stories.
- The user stories that will be finished in the next release will be selected.
- Based on the user stories and the estimates, the release date is determined.

Active listening is significant in this phase, as-

- The developers need to understand what functionality they need to code for the current release and the effort and duration (estimates) required to deliver this functionality.

- The customer and the developers need to understand the feasibility for commitment on the date of next release.

Release Planning – Steering Phase

In Steering phase the customer and developers "steer" -

- To make changes in individual user stories and the relative priorities of different user stories.

- To adjust the plan.

- If estimates were proved wrong.

- To accommodate the changes.

Active listening is significant in this phase,

- To understand-
  - The new requirements to be added.
  - What changes needs to be done for existing requirements.
  - The impact on the existing system if any of the existing requirements is removed.
- Arrive at the estimates required to adjust the plan, considering
  - The work done so far.
  - The new requirements to be added.
  - Existing requirements that have to be changed or removed.

Iteration Planning

In Iteration Planning, the developers are involved to plan the activities and tasks for the iteration. In this, the customer is not involved.

Iteration Planning has three phases-

- Exploration phase.
- Commitment phase.
- Steering phase.

Iteration Planning – Exploration Phase

In Exploration phase,

- Requirements will be translated to different tasks.
- The tasks are recorded on task cards.
- The developers estimate the time it will take to implement each task.
- If the developers cannot estimate a task because it is too small or too big, they need to combine or split the task.

Iteration Planning - Commitment Phase

In Commitment phase,

- The tasks is assigned to the developers.
- The developer accepts the tasks for which he or she takes responsibility.
- The developer estimates the time it takes to complete the task because the developer is now responsible for the task, and he or she should give the eventual estimation of the task.
- Load balancing is done after all the developers within the team have been assigned tasks.
- A comparison is made between the estimated time of the tasks and the load factor.
- The load factor represents the ideal amount of hands-on development time per developer within one iteration assuming a 40-hour week.
- Then the tasks are balanced out among the developers. If a developer is overcommitted, other developers must take over some of his or her Tasks and vice versa.

Iteration Planning – Steering Phase

In Steering phase,

- The developer gets the task card for one of the tasks to which he or she has committed.
- The developer will implement this task along with another developer following the pair programming practice.

Implementation

The implementation of the tasks is done. The implementation includes design, writing unit tests, coding, unit testing, refactoring, continuous integration to the code base, integration testing and acceptance testing based on the task card and user story card.

Extreme Programming Artifacts

Extreme Programming is not anti-documentation, but encourages doing the least amount that is really needed. The document when needed for distributed sharing, historical needs, summarizing, etc.

The essential Extreme Programming artifacts are-

- User story cards
- Acceptance tests
- Estimates
- Release plan
- Iteration plan
- Task cards
- Design
- Unit test cases
- Customer and developer communication records

User Story Cards

The User story cards have the following features-

- A user story card-
  - Contains a short description of the behavior of the system from the point of view of the customer.
  - Must be written by the customer and not by the developers.
  - Uses the customer's terminology without technical jargon.
  - Should only provide enough detail to make an estimate of how long the story will take to implement.
- One for each major feature in the system.
- Are used to create time estimates for release planning.
- Drive the creation of the acceptance tests.

Acceptance Tests

Acceptance tests must be one or more tests to verify that a story has been properly implemented.

Estimates – Release Planning

Effort and duration estimates for the stories that will be used in release planning-

- Arrive at the target release date in exploration phase.
- Plan adjustments in steering phase.

Release Plan

Release plan contains –

- The user stories selected for the release.
- Effort and duration estimates.
- Target release date that is committed.

Task Cards

A Task card –

- Contains the required tasks to implement a user story.
- One task card per user story.
- Forms basis for task estimates and task assignments.

Estimates – Iteration Planning

Effort and duration estimates for the tasks based on the user stories are evaluated that will be used in iteration planning for the task assignments and load balancing in commitment phase.

Iteration Plan

Iteration plan contains –

- The user stories selected for the iteration
- Task assignments
- Task estimates

Design

A design contains a simple design just enough for the implementation of a user story.

Unit Test Cases

This contains the Unit Test Cases that drive the coding and unit testing.

Customer and Developer Communication Records

The customer and developers discuss the story to elaborate on the details. It is verbal when possible, but documented when required.

10. XP – Rules

Consider any sport that you play. You need to abide by the rules of that sport or game. In a similar way, in Extreme Programming as the entire project is driven by collaboration among the team members and with the business (who represents the customer), certain rules for the project need to be laid out at the beginning itself. These rules should be at par with the Extreme Programming practices.

The rules provide a common reference for everyone on the team so that they remind everyone of what and how they need to do when things go smoothly and also when they are not going well.

The game that we refer to in the Extreme Programming is Planning Game.

Rules of Planning Game

In Extreme Programming, the planning game begins with the first-release planning meeting and ends with the final release.

You must define the rules of the Planning Game in line with the Extreme Programming practices before the first release planning meeting and familiarize the rules to the business and the team. You must manage the project ensuring that the rules are adhered to.

However, in software development, there is no way that a simple set of rules apply in every project. They may have to vary according to the business and the team, not compromising on the Extreme Programming practices. Hence, a set of rules with the necessary goals can initially be put in place and they can be modified as the development progresses only if required.

The goal of the game is to maximize the value of software produced by the team. From the value of the software, you have to deduct the cost of its development, and the risk incurred during development.

The strategy for the team is to invest as little as possible to put the most valuable functionality into production as quickly as possible in conjunction with the design and coding strategies to reduce risk.

In view of the technology and business lessons of this first working system, it becomes clear to business what is now the most valuable functionality, and the team quickly puts this into production.

This process continues with iterations and releases till the entire development is complete.

The basic rules of planning game can be categorized into the following areas-

- Planning
- Managing
- Designing
- Coding
- Testing

Planning

Planning is done during release planning and iteration planning. The rules are same for both.

In Release planning,

- Business and the team are the players.
- Story cards are used.
- User stories are written by the customer on story cards.
- Business is decided on the priority of the functionality for implementation.
- Estimates are given by the team based on the story cards.
- Short (frequent small) releases are to be planned
- Release schedule is to be created with mutual agreement.
- The next release is to be divided into iterations.

Iteration planning starts each iteration.

In Iteration planning,

- The team members are the players.
- Task cards are used.
- For each story selected for the iteration, the task cards are produced.
- The team members have to estimate the tasks based on the task cards.
- Each team member is assigned with task cards.
- Each team member has to then re-estimate based on his or her assignment, for
    - Accepting the work.
    - Taking responsibility of the completion of the work.
    - Getting feedback about the actual time taken.
    - Improving the estimates.
    - Respecting those estimates.

Thus, the rules for who can make and change the estimates are clear.

- Final assignment is done assuming 40-hour week and load factor.

Managing

- The team is given a dedicated open workspace.
- Each workstation is to be arranged such that two developers can sit side by side and easily slide the keyboard and mouse.
- Sustainable pace is to be set (40-hour week and no overtime for more than one week) and managed.

- Enforce the rules of the planning game.
- Fixing any extreme programming practice when it breaks.
- Ensure Communication among the Team.
- Discourage Communication that is-
  - not helpful
  - not at the right time
  - done in great detail
- Make people move around.
- Measure the actual times and convey to team periodically so that each team member will know the performance as against prediction. This ensures that the team member improves in estimating.
- Make food available to the team as and when required.

Designing

The rules of designing are-

- Choose a metaphor for the system and evolve it as development progresses.
- Keep the design simple.
- No functionality is added early.
- Put as much architecture in place now as you need to meet your current needs, and trust that you can put more in later
- Refactor whenever and wherever possible.

Coding

The rules of coding are-

- The business (who represents the customer) should always be available.
- Developers should write all the code in accordance with rules emphasizing communication through the code.
- Pair programming should be ensured.
- Coding standards are to be followed.
- All code must have unit tests.
- Write a unit test first, before the code is written for each piece of the system so that it is much easier and faster to create your code. The combined time it takes

to create a unit test and create some code to make it pass is about the same as just coding it up straight away. It eases regression testing.

- When you are coding you should be wearing only one of the following four hats-
  - Adding new functionality, but only changing the implementation.
  - Adding new functionality, but only changing the interface .
  - Refactoring code, but only changing the implementation.
  - Refactoring code, but only changing the interface .
- A dedicated integration workstation is provided for the entire team.
- Only one pair integrates code at a time and ensures all the tests are passed.
- Integration should be done often.
- Collective ownership should be used.

Testing

The rules of testing are-

- All codes must pass all unit tests before it is released.
- Tests are to be written when a defect is found.
- Acceptance tests are to be run often.

11. XP – Additional Features

In this chapter, we will learn about some additional features of Extreme Programming.

Feedback Loops

The charm of Extreme Programming is continuous feedback that keeps everyone focused and development continues in the right direction without any delays.

In Extreme Programming, feedback is accomplished at different levels, to the required and sufficient detail. This is done continuously and constantly across the iterations and releases as well.

The following table illustrates the feedback events and the feedback duration.

Feedback Event	Feedback Duration
Pair Programming	Seconds
Unit Testing	Minutes
Clarifications among Team and with the Customer	Hours
Progress	In a Day
Acceptance Testing	Days
Iteration Planning	Weeks
Release Planning	Months

Project Management

In Extreme Programming, Project Management is not given emphasis and the manager role performs the minimal and the most essential management activities.

However, these activities embed the project management requirements.

- Release Planning
  - Scope is defined by user stories in story cards.
  - Estimates are story estimates, that can be in story points.
  - Delivery milestones are captured by release plans.
  - Release is divided into iterations.
- Iteration Planning
  - Stories are blown into tasks in task cards.
  - Estimates are task estimates.

  - Allocation of tasks done by balancing load factor across the team.
  - Task acceptance is by team commitment and accountability.

- Tracking
  - Velocity in terms of story points from actual time for implementation done at project level.
  - Productivity in terms of task estimates from actual time for implementation done at developer level.
  - Defect tracking.

Extreme Programming – Industry Experiences

From the Extreme Programming projects executed across the industry, there are certain learnings useful for the teams.

Learnings from the XP Practices

In the following sections, you get an understanding of these learnings.

- **Simple Design:** A simple design is efficient, easy to build and maintain.
- **Metaphor:** The main intention of using metaphor is to use domain specific names throughout development and that enables the customer also to actively participate in the development.
- **Collective Ownership:** Each one is proud of their good code. By working together, each one is proud of everyone's code.
- **Planning:** In case the team completes many user stories in an iteration, make the iterations smaller. Have the team consensus to alter a plan.

Scaling Extreme Programming

Initially, Extreme Programming was perceived to be effective in smaller teams, with a team size up to 12-16 developers.

Later, it was observed that it is possible to scale Extreme Programming up to teams of 40-50. However, it is recommended to do the scaling by building recursive teams. Build a small team first, grow the team incrementally, using the first team to seed recursive teams.

Documentation

Extreme Programming is not against any documentation. It encourages documenting what is required, when it is required and only to the required and sufficient Detail.

This may differ from project to project and it is for the project to decide on the extent of documentation. However, skipping documentation is not allowed by the Extreme Programming practices.

Advantages of applying XP

Extreme Programming is found to be advantageous when applied in-

- Environments that are highly uncertain
- Internal projects
- Joint ventures

Disdvantages of applying XP

Extreme Programming is found to be disadvantageous when-

- The environments are large and complex.
- The requirements are well understood .
- The customer is at a distance or unavailable.

Misconceptions of XP

There are some misconceptions about Extreme Programming. The following table gives the clarifications of the misconceptions that exist.

Misconception	Clarification
No written documentation	Minimal and sufficient documentation needs to be there. However, there are no formal standards for how much or what kind of documents are needed.
No design	Minimal explicit and up-front design needs to be there that evolves as the development progresses.
Extreme Programming is just pair programming and easy	Pair Programming is just one of the Extreme Programming practices. It requires great discipline and consistency that is achieved in conjunction with the other Extreme Programming practices.
Extreme Programming is the one, true way to build software	Extreme Programming is effective only in certain type of projects.

12. XP – Scrum + Extreme Programming

Extreme Programming is one of the earliest agile methodologies that came into existence and is continuously evolving. Kent Beck, who evolved Extreme Programming, developed it with the premise to use best programming practices and take them to the extreme.

Extreme Programming in the current scenario focuses on the best practices that are prevailing in the industry as of now and takes them to the extreme. The most prevalent examples are Test Driven Development, Whole Team Approach, Sustainable Pace, etc.

XP – Flexibility as Technique

One of the reasons Extreme Programming is so popular is its flexible nature. Further, Extreme Programming is more about technique than process and hence merges well with the process-centric approaches. So, you can easily combine features of extreme programming with other ideas, wherever

- Some of the Extreme Programming practices are complimentary in the process.
- Extreme Programming by itself is not suitable for implementation.

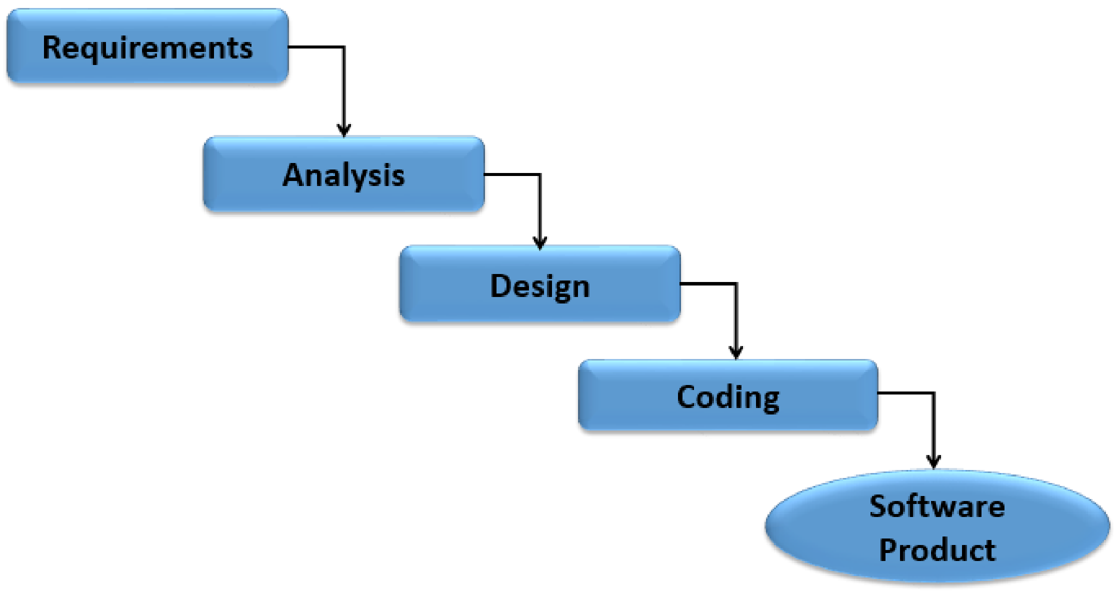
However, remember that any of the Extreme Programming Practices that you choose should be implemented to its core. Otherwise, you cannot claim that you are using Extreme Programming.

In fact, this is true for any process that you take up. Mix-and-match is allowed, but only when you are using complimentary features and you are not compromising on the values of the features being used.

The most popular Extreme Programming hybrid that is currently in use is Scrum + Extreme Programming hybrid. We will start with the basic and still prevalent Software Development Methodology – Waterfall Model.

Waterfall Model

In Waterfall model, the development progresses in phases and no phase can begin before the completion of the earlier phase.



Waterfall Model is still in use in several organizations though some perceive it as a traditional methodology. It is an established and effective methodology if the requirements are completely known before the development starts. The process is straightforward and does not require any training or mentoring.

What you need to remember is that no methodology is ideal for every situation and every methodology will have its own merits and demerits. Hence, you have to understand which methodology suits your context, your environment and your customer interests.

Let us have a look at the shortcomings of the Waterfall methodology-

- As all the requirements are to be known before the initiation of development, it is not suitable when the requirements are incomplete or vague.

- As the methodology is forward facing in one direction, feedback from any phase cannot be taken back into any of the earlier phases, though more clarity is obtained regarding the end product.

- Testing is taken up only after the completion of development, with a team of developers who were not involved in the earlier phases, such as requirements gathering or development. This test-last approach often leads to defect containment, high defect rates, high delivered defects that are uncovered by the customer.

- The working product will not be available till the end of development and hence nobody will have a clue on whether the right thing is being built though it is being developed rightly.

- Defect fixes and changes to the requirements are difficult to be absorbed as there is a high chance of breaking the design and also the costs incurred are high.

If you predict such conditions in your development, the Agile methodologies are the most suited.

Agile Methodologies

Agile Methodologies advocate-

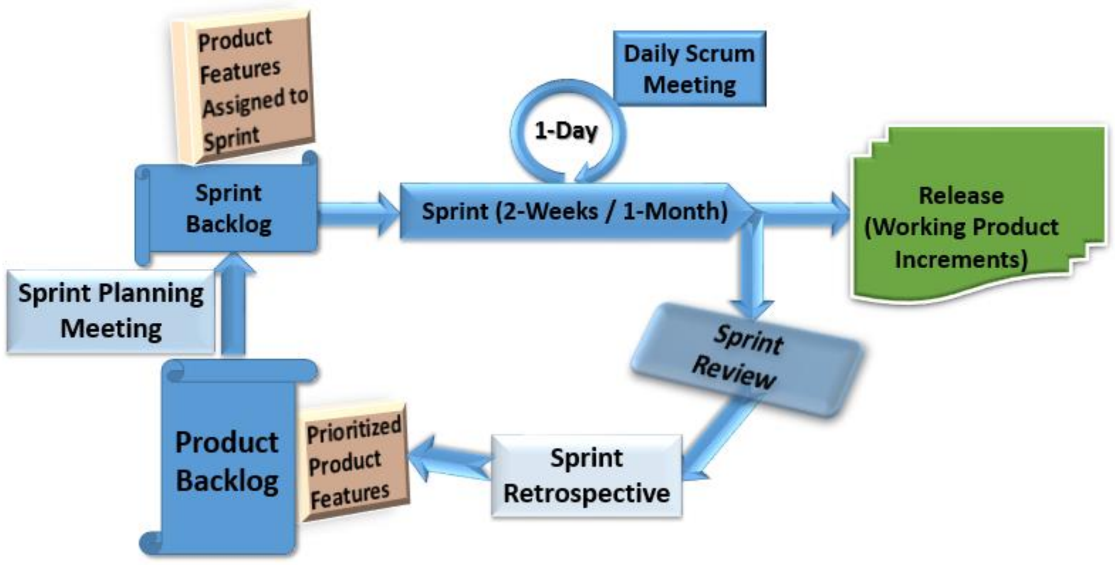
- Frequent releases of working product increments, enabling feedback to flow in at right time.
- Team-centric approach to make everyone responsible and accountable for the end product.
- Flexible planning that focuses on the delivery of value to the customer, meeting customer expectations, return on investment.
- Readiness to accept changes at any point of development so that the end product that is delivered is not obsolete.

Several Agile methodologies came into existence, of which Scrum has become more popular and is being widely used.

How Scrum makes the Difference?

In Scrum, projects are broken down into releases and time-boxed short sprints. For every sprint, you will take up only the required and sufficient functionality that is prioritized by the customer and that you can deliver in a sprint. At the end of each Sprint, you will have a working product that could be released.

The requirements are called backlog items, the iterations are called sprints. Continuous Testing with test-first approach will be adapted. The developers and the testers also participate in the user story writing which are the backlog items. This gives an upfront view of the product's behavior to everyone in the team and also helps in arriving at the acceptance criteria at the beginning of the sprint itself.


Product Features Assigned to Sprint
Daily Scrum Meeting
1-Day
Sprint Backlog
Sprint (2-Weeks / 1-Month)
Release (Working Product Increments)
Sprint Planning Meeting
Sprint Review
Product Backlog
Prioritized Product Features
Sprint Retrospective

Scrum, by its definition, again as we stated earlier, is effective in certain situations, but has its own shortcomings as with any other development methodologies.

- The time-boxed sprints will not allow any flexibility in the release schedule that hampers both development and testing.

- Scrum, on its own do not give directions for development.

Hence, Scrum is usually combined with other Agile methodologies that focus more on the development strategies.

Scrum + Extreme Programming Hybrid

Scrum is being used quite frequently incorporating Extreme Programming practices that are complimentary, with Extreme Programming focusing on the engineering aspects such as continuous communication, frequent feedback loops, refactoring, collective ownership, continuous integration, test-driven development, etc. and scrum focusing on the fixed scope for sprints, burn-down charts, etc.

- As Scrum is a defined methodology, it is easier to adapt from day-one of the project.

- As Extreme Programming is more towards communication and team cohesion, the team is more focused on the development.

Thus, a Scrum + Extreme Programming hybrid is found to be effective.

Tools for Scrum + XP Hybrid Projects

Tools such as SpiraTeam and Rapise are meant for Scrum + Extreme Programming Hybrid Projects.

SpiraTeam provides reporting dashboards of key project quality and progress indicators in one consolidated view that is tailor-made for Scrum and Extreme Programming projects.

Some of the indicators are-

- Requirements Test Coverage
- Task Progress
- Project Velocity
- Top Risks and Issues

Rapise tool is a test automation solution that can be integrated fully into your development process and that be adapted to your changing needs. You can use it for testing desktop, web and mobile applications. This tool can be used by developers, testers and business users to generate acceptance tests.

13. XP – Tools

In this chapter, we will learn about some tools used in Extreme Programming.

ExtremePlanner

ExtremePlanner is a browser-based Agile Project management solution that is designed specifically to support Agile methods including Scrum and Extreme Programming.

ExtremePlanner concentrates on planning and tracking the progress of features (or user stories) that have actual business value to customers.

The key features of ExtremePlanner are-

- Supports the whole team including project managers, developers, QA, tech support, and stakeholders.
- Estimates and plans software releases with drag and drop ease.
- Manages features, defects, test cases and development tasks in one place.
- Has an integrated issue tracking to manage customer requests from start to finish.
- Provided the latest changes with email notifications and project activity reports.

For more Information: http://www.extremeplanner.com/

Project Planning and Tracking System

PPTS is a Web-based environment supporting teams who have chosen to develop Software according to the Agile Methodologies Scrum and/or Extreme Programming.

PPTS functionality includes-

- Administration of project, iteration and resource attributes
- Product backlog that can be prioritized
- Work breakdown structure (sprint backlog)
- Metrics (velocity and estimated/spent effort)
- Burndown and progress charts
- Calendars
- Resource allocation

- Fine grained access to information based on overall role (administrator or user) or role in project (project leader, developer or customer)
- Customization of menus and language (both English and Dutch available)
- Interfacing with PR/CR tools

For more Information: http://ses-ppts.sourceforge.net/

Targetprocess

Targetprocess is a Visual project management software that enables you to visually manage complex work and focus on the things that matter.

Targetprocess gives the visibility and transparency you need across your organization. From Kanban and Scrum to almost any operational process, Targetprocess flexibly adapts to your management approach and organizational structure.

Targetprocess provides-

- Boards for planning and progress tracking. Board view gives many options to handle a large amount of cards seamlessly.
- Boards that can be shared with any person to broadcast information outside. They are flexible.
- Multiple cards can be moved with drag and drop.
- Lists for project hierarchy and manage backlogs with ease.
- Full customization, inline edit and beautiful design.
- Graphical reports.
- Timelines.
- Custom views.
- Dashboards.

For more information: https://www.targetprocess.com/

Plone Extreme Management Tool

Plone Extreme Management tool provides project administration which supports the Extreme Programming methodology.

Plone Extreme Management tool provides-

- Content types-
    - **Project:** Project Managers can add multiple projects. For each project, iterations and stories can be added by both the customers and employees.
    - **Iteration:** The project will be planned with iterations. An iteration is a period of 1 to 3 weeks in which a number of stories will be implemented.
    - **Offer:** Contains the stories that a customer wants in this project. It is used as a way to bundle the wishes of the client and give a first indication of the size of a project.
    - **Story:** The customer can define new features by describing these feature in a story.
    - **Task:** The employees can estimate the story by defining tasks.
    - **Booking:** While working on tasks the employees can track time and easily book those at the end of the day.
- Workflow.
- Time tracker.
- Release plan.
- Iteration roundup.

XP Tools for Java Developers

The following table gives a list of tools for Java developers for related activities.

Java Extreme Programming Tools	Activity
Maven and AntHill	Project management and continuous integration.
Ant and XDoclet	Automated building and continuous integration.
AntHill and CruiseControl	Automating continuous integration .
IntelliJ Idea, Xrefactory, DPT, Jfactor, Jrefactory	Java refactoring.
JUnit	Automated Java testing.

Cactus	Automated Servlet, JSP, and other J2EE testing.
Jemmy, JFCUnit, and Abbot	Automated swing testing.

XP Tools for .Net Developers

In lines with Java, .Net has NAnt, NUnit, CruiseControl.NET. Visual Studio has many refactoring tools.

Adopting XP in your Organization

If you are planning to adopt Extreme Programming in your organization, first you select a project suitable for Extreme Programming and a team. Get an experienced coach. Get the team accustomed to the Extreme Programming practices, estimation and team communication.

Initiate the project with the minimal essential Extreme Programming rules for the project. Allow the rules to evolve for better implementation. Take into consideration the synergies across the Extreme Programming practices. Allow sufficient time for the team to scale the learning curve. Manage the team culture and change.

It is advised to take an internal project at first. Once you successfully implement that project, you will have the team and the management to stand by you for the extension to other suitable projects.