eXtreme Programming

XP

- Done by :

Mohammed Gamal

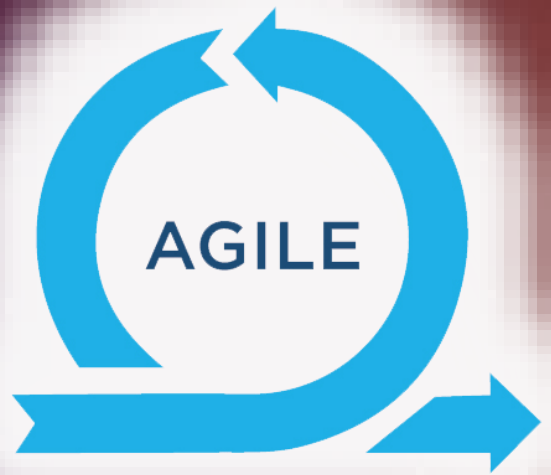
الجامعة الإسلامية بالمدينة المنورة

ISLAMIC UNIVERSITY OF MADINAH

Contents

- Agile Methodology
- eXtreme Programming (XP)
- When to use XP
- Values & Principles
- Practices

Agile Methodology



Agile software development is based on fundamental changes to what we considered essential to software development ten years ago.

The most important thing to know about Agile methods or processes is that there is no such thing. There are only Agile teams. The processes we describe as Agile are environments for a team to learn how to be Agile.

Extreme Programming (XP)

Definition:

Extreme programming or XP is a software development methodology (Type of agile software development) which is intended to improve software quality and responsiveness to **changing customer requirements**. it produce frequent *Releases* in short development cycles, to improve productivity

Extreme Programming (XP)

P Creator:

Kent Beck

American software engineer and the creator of Extreme Programming, He was born in 1961

He created XP during his work on the C3 project.

He became the C3 project leader in 1996 and began to refine the development methodology sed in the project

Extreme Programming (XP)

XP Properties:

- **lightweight:** because XP does not overburden the developers with giant processes.
- **efficient:** XP always seeks to getting the best result with minimum cost.
- **low-risk:** Strategies followed in XP minimizes the chances of system failure.
- **flexible:** because it very friendly with frequently changing requirements.
- **predictable:** because everything within XP is follows a plan.

Extreme Programming (XP)

“Extreme” Meaning:

Extreme means that take the effective principles and practices to extreme levels.

- **Code Review** is effective as the code is reviewed all the time.
- **Testing** is effective as there is continuous testing.
- **Design** is effective as everybody needs to do refactoring daily.
- **Iterations** are effective as the planning game for release planning and iteration planning.

Where to use XP?

- Customer cannot describe the system.
- Requirements change rapidly and frequently
- Low Technical Skills
- Small Team
- High Risk
  - New Technology
  - Unclear requirements
  - Limited Time

XP Values & Principles

XP provides values and principles to guide team behavior.
The team is expected to be self-organize.

- **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.

Note:
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’.

The principles here are:

- Do what is needed and asked for, but no more.
- Do the simplest thing that works
- Take small simple steps to your goal
- Never implement a feature you do not need now.

Feedback

Feedback is very important value in XP because it helps to drive changes.

it occurs at different levels and phases in the process of the development, for example:

- **Unit Tests:** Unit tests tell the developers the status of the system.
- **User Stories:** Developers estimate new stories as soon as they get them from the customer.
- **Frequent Releases:** These enable the customer to perform acceptance tests and provide feedback and developers to work based on that feedback.

Courage

Courage means developer must possess the courage to…

- Throw away code if it doesn't work
- Change it if you find a way to improve it
- Fix it if you find a problem
- Try out new things if you think that they might work better than what you have right now.
- Communicate and accept feedback
- Tell the truth about progress and estimates

Respect

- Everyone respects each other as a valued team member.
- 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.

Practices

XP provides specific core practices where each practice is simple and self- complete.

- **Management Practices:**
  - On-Site Customer
  - Incremental Planning
  - Small/Short releases

- **Team Practices:**
  - Metaphor
  - Collective Ownership
  - Continuous Integration
  - Coding Standards
  - Sustainable Pace

- **Programming Practices:**
  - Test First
  - Simple Design
  - Refactoring
  - Pair Programming

P1 (On-Site Customer)

The customer is an actual member of the team so the customer will

- Sit with the team
- Bring requirements to the team
- Discuss the requirements with them

There is a common objection to this practice is the fact that it is almost impossible in the real world to find a customer that ready to staying with the team all the time

The answer to that objection is that if the system is not worth the time of one customer then maybe the system is not worth building. In other words if you're investing big amount of money in building a system you might just as well invest a little more and have one of the people in the customers organization stay with the team and be involved in the whole process.

P2 (Planning Game)

The main planning process within XP is called the Planning Game. It is a meeting that occurs once per iteration.
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	Technical people
Scope: How much of a problem must be solved for the system to be valuable in production? **Priority:** If you are given an option, which one do you want? **Dates of releases:** What are important dates at which the presence of the software would make a big difference?	**Estimates:** How long will a feature take to implement? **Process:** How will the work and the team be organized? **Detailed Scheduling:** which stories should be done first?

P3 (Small/Short Release)

We should start with producing a simple version of system 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

P4 (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"
"the city is a jungle"

You should guide the entire development with a simple shared story of how the whole system works. So, XP suggests to try to design a system that is easy to explain using real-life analogies.

P4 (Metaphor)

For example:

Project	Metaphor	Explanation
Variable transparency window	Chameleon	The window will use nanotechnology to vary opaqueness depending on senor readings, e.g. temperature. It can also generate decorative patterns. This is the architecture project.
Ford Motor Company software architecture tool	Compiler	A tool is being developed to combine architectures of components into one major architectural artifact.

P5 (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.

P6 (Continuous Integration)

Code is integrated and tested many times a day. A simple way to do this is to have a machine dedicated to integration.

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

P7 (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).

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.

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.

P8 (40-Hour Week)

XP emphasizes on the limited number of hours of work per week for every team-members, 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.

P9 (Testing)

Testing is a very importing aspect in XP, where some times the test will be written before the code is written.

We have many types of testing, but in fact we interesting in 2 in particular in XP:

- **Acceptance Test**
- **Unit Test**

Other Types of Tests:

- Integration testing
- System testing

P9 (Testing)

1- Acceptance Test:

Also known as Functional testing. Acceptance tests are black box system tests. That means we ignores the internal parts, and focus on the output is as per requirement or not.

Each acceptance test represents some expected result from the system. Customers are responsible for verifying the correctness of the acceptance tests and reviewing test scores to decide which failed tests are of highest priority.

Acceptance tests are created from User Stories. During an iteration the User Stories selected during the Iteration Planning Meeting will be translated into acceptance tests. A story can have one or many acceptance tests, whatever it takes to ensure the functionality works.

P9 (Testing)

2- Unit Test:

Unit Test in general is a test that testing individual software components or modules. Typically done by the programmer and not the testers, as it requires details knowledge of the internal program design and code.

Unit test is one of the corner stones of Extreme Programming (XP). But unit tests XP style is a little different, for example:
The test will be created before the code.
We should test all classes in the system. (getter and setter methods are usually omitted).

P10 (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
- Has the fewest possible classes and methods

P11 (Refactoring)

Refactoring is a way of improving the system. So the developer ask questions like:

- How to make the code simpler.
- Is there any way to restructure the system without changing its behavior to remove duplication, improve communication, simplify, or add flexibility.

Advantages:

- improve the product as a whole
- Increases the developer knowledge of the system

P12 (Pair Programming)

Pair Programming is a method where two programmers are working using one computer sitting side by side.

In pair programming we have two role:

- One is the **Driver** (or the coder): the one who is writing the code.
- The other is the **Observer** (or the navigator) which is the one who reviews each line of code as it is typed in.

The two programmers switch roles frequently.

Pair Programming if done well means less mistakes are made and should cost the business less.

Advantages:

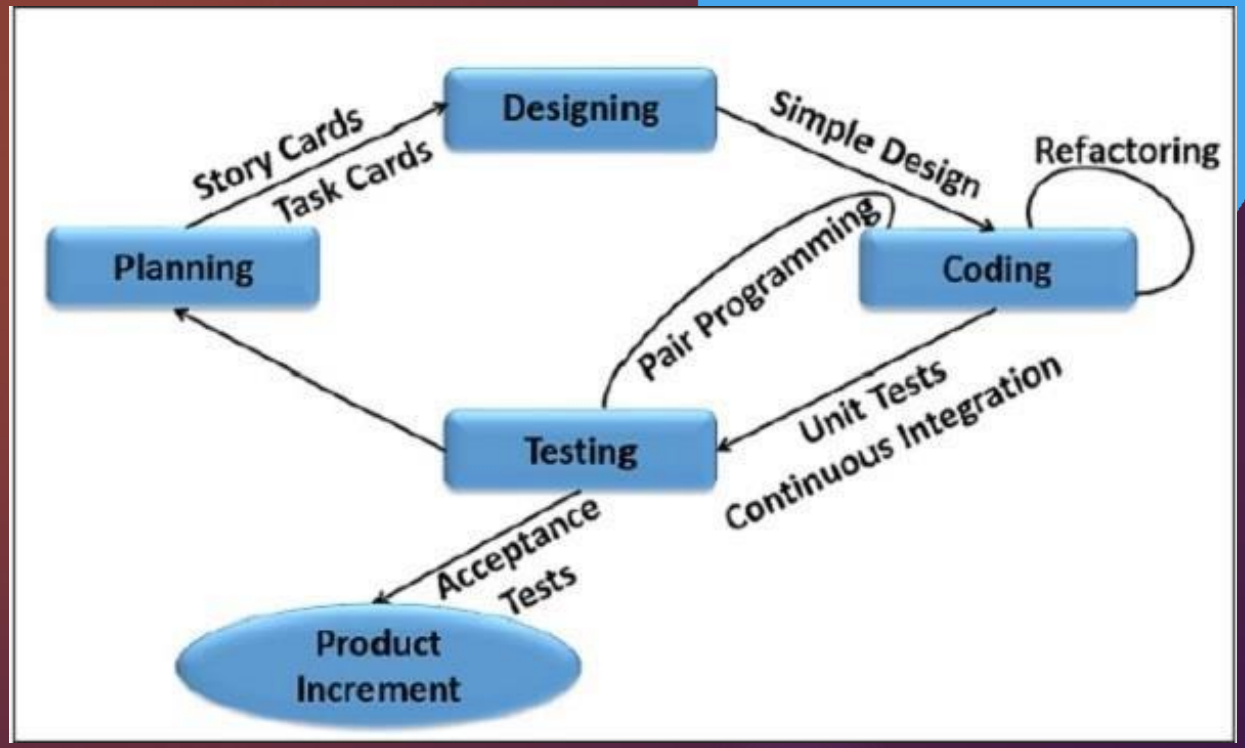
- Roles between the two programmer are switched resulting in emergence of new ideas.
- Programs with fewer bugs.
- Post-production maintenance cost is much less.
- Programmers learn from each other.

XP Basic Activities

XP activities need to be structured in light of the XP principles. This why XP practices are defined:

XP consist of four basic activities. They are:

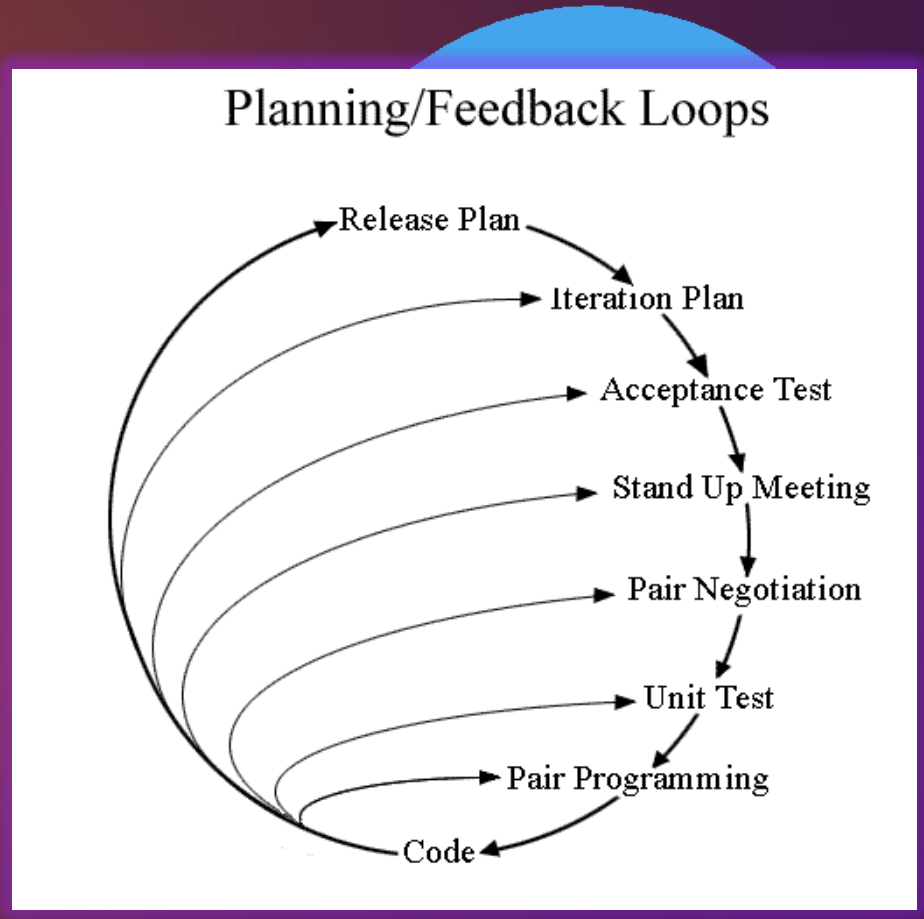
- Planning
- Designing
- Coding
- Testing



XP Cycles

Incremental planning is based on the idea that our requirements are recorded on story cards, sort of use cases or scenarios that the customer provides. So, the first step in **Incremental Planning** is:

1. Select User Story
2. Release Plan
3. Iterative Plane
4. Development Process
5. Release Software
6. Evaluate System and Iteration



1. Select User Story

User Stories are short, simple descriptions of a feature told from the perspective of the person who desires the new capability, usually a user or customer of the system. They typically follow a simple template:

As a < type of user >, I want < some goal > so that < some reason >.

1st Step is selecting *User Stories* for a given release and determine which stories exactly to be included taking into account how much time available and on the priority (priority of a *User Story* is determined according to their importance to the user).

2. Release Plan

After *User Stories* have been written team can start the make a *Release Plan*. The *Release Plan* specifies which *User Stories* are going to be implemented for each system *Release*, and dates for those *Releases*. Also, The *Release Plan* sets the *Release* goal and Release time-frame (This gives the customers a set of *User Stories* to choose from during the ***Iteration Planning Meeting*** to be implemented during the next *Iteration*).

Note: An ***Iteration Planning*** is done after ***Release Planning.***

3. Iterative Plane

After *Plan Release* the *User Stories* are selected (the selected *User Stories* for an iteration is done by the customer from the *Release Plan*) After that the **3rd Step** comes in which is *Iterative Plan*. One of the main tasks that performed in *Iterative Plan* is breaking selected *User Stories* into individual programming tasks to be implemented during the Iteration. So, we take the *User Stories* and we identify specific development tasks that we need to perform in order to realize these Stories.

Note that: An ***Iteration Planning Meeting*** is performed at the beginning of each *Iteration* to produce that *Iteration's plan*.

In **Iteration Plan** the team:
Breaks down the *User Stories* into independent tasks (around 8 hours each).
Performs Risk Assessment of stories and decides on a lightweight plan on how to address the risks as the project moves forward.
Creating acceptance tests for user stories (which is in next step).

4. Development Process

After finishing all these steps and determine and define our software releases, what user stories will be taken in account in each release, how many iterations, what user stories tasks will be done in each iteration, and so on and so forth … etc. it is the time to start performing our planning steps. So, here we can start develop, integrate and test the code

Of course, these processes (Develop, Integrate and test) are iterative processes (each will be done many times).

4. Development Process (Develop)

Developers sign up to do the tasks and then estimate how long their own tasks will take to complete.

Note that: It is important for the developer who accepts a task to also be the one who estimates how long it will take to finish. People are not interchangeable and the person who is going to do the task must estimate how long it will take. The ideal development time is how long it would take to implement the Story in code.

if there were no distractions, no other assignments, and you knew exactly what to do. Longer than 3 weeks means you need to break the story down further. Less than 1 week and you are at too detailed a level, combine some stories. About 80 user stories plus (or minus) 20 is a perfect number to create a release plan during release planning.

4. Development Process (Integrate)

Intergrade means ones we finish a new feature or functionality we should add to the previous parts, until eventually we get the whole system.

This is why we have integration testing which is testing of integrated modules to verify combined functionality after integration.

Example

Quiz

Which of the following statements about extreme programming are true?

[ ] Because of pair programming XP requires twice the number of developers
[ ] In XP code is rarely changed after being written
[ ] The customer does not need to provide any requirements in XP
[ ] XP is an iterative software development process

Quiz

Which of the following statements about extreme programming are true?

[F] Because of pair programming XP requires twice the number of developers
[ ] In XP code is rarely changed after being written
[ ] The customer does not need to provide any requirements in XP
[ ] XP is an iterative software development process

Quiz

Which of the following statements about extreme programming are true?

[F] Because of pair programming XP requires twice the number of developers
[F] In XP code is rarely changed after being written
[ ] The customer does not need to provide any requirements in XP
[ ] XP is an iterative software development process

Quiz

Which of the following statements about extreme programming are true?

[F] Because of pair programming XP requires twice the number of developers
[F] In XP code is rarely changed after being written
[F] The customer does not need to provide any requirements in XP
[ ] XP is an iterative software development process

Quiz

Which of the following statements about extreme programming are true?

[F] Because of pair programming XP requires twice the number of developers
[F] In XP code is rarely changed after being written
[F] The customer does not need to provide any requirements in XP
[T] XP is an iterative software development process

Thank You