Chapter -1

Agile Development

What is Agile?

- Agile is the ability to create and respond to change.
- It is a way of dealing with, and ultimately succeeding in, an uncertain and turbulent environment.
- The authors of the Agile Manifesto chose "Agile" as the label for this whole idea because that word <u>represented the adaptiveness and response to change which was so important to their approach</u>.

Agile Software Development

- **Agile software development** is a conceptual framework for software engineering that promotes development iterations throughout the life-cycle of the project.
- Software developed during one unit of time is referred to as an iteration, which may last from one to four weeks.
- Agile methods also emphasize working software as the primary measure of progress

Agile Practices

- Many of us have lived through the nightmare of a project with no practices to guide it. The lack of effective practices leads to unpredictability, repeated error, and wasted effort. Customers are disappointed by slipping schedules, growing budgets, and poor quality.
- Developers are disheartened by working ever-longer hours to produce ever-poorer software.
- Once we have experienced such a fiasco, we become afraid of repeating the experience.
- Our fears motivate us to create a process that constrains our activities and demands certain outputs and artifacts.
- We draw these constraints and outputs from past experience, choosing things that appeared to work well in previous projects. Our hope is that they will work again and take away our fears.
- But projects are not so simple that a few constraints and artifacts can reliably prevent error.

Agile Practices

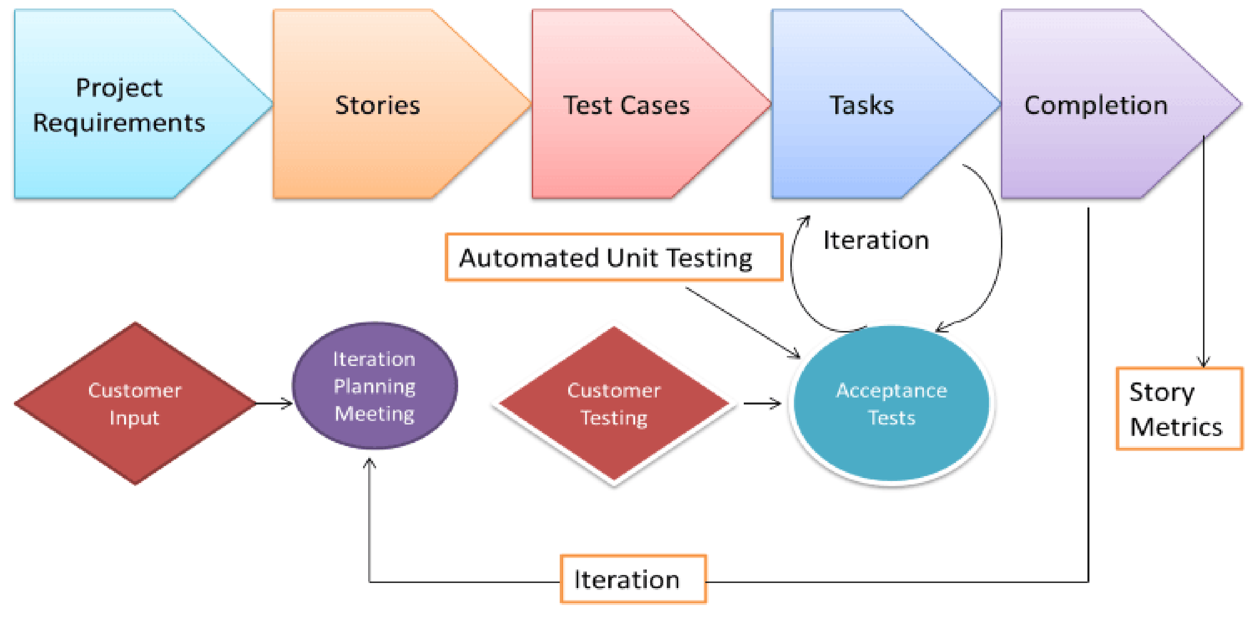
- As errors continue to be made, we diagnose those errors and put in place even more constraints and artifacts in order to prevent those errors in the future. After many projects, we may find ourselves
- overloaded with a huge, cumbersome process that greatly impedes our ability to get projects done.
- A big, cumbersome process can create the very problems that it is designed to prevent. It can slow
- the team to the extent that schedules slip and budgets bloat. It can reduce the responsiveness of the
- team to the point of always creating the wrong product. Unfortunately, this leads many teams to
- believe that they don't have enough process. So, in a kind of runaway process inflation, they make
- their process ever larger.
- Runaway process inflation is a good description of the state of affairs in many software companies
- circa 2000. Although many teams were still operating without a process, the adoption of very large,
- heavyweight processes was rapidly growing, especially in large corporations.

What is Extreme Programming (XP)

- Extreme Programming (XP) is an agile software development framework that aims to produce higher quality software, and higher quality of life for the development team.
- XP is the most specific of the agile frameworks regarding appropriate engineering practices for software development.

What is Extreme Programming (XP)

- Extreme Programming technique is very helpful when there is constantly changing demands or requirements from the customers or when they are not sure about the functionality of the system.
- It advocates frequent "releases" of the product in short development cycles, which inherently improves the productivity of the system and also introduces a checkpoint where any customer requirements can be easily implemented.
- The XP develops software keeping customer in the target.



Extreme Programming (XP)

Continue…

- Business requirements are gathered in terms of stories. All those stories are stored in a place called the parking lot.

- In this type of methodology, releases are based on the shorter cycles called Iterations with span of 14 days time period. Each iteration includes phases like coding, unit testing and system testing where at each phase some minor or major functionality will be built in the application.

Why Extreme Programming (XP)?

- Extreme Programming works towards providing iterative and recurrent software releases throughout the project; instead of everything together after a single, long project development lifecycles.

- These short iterative cycles help both team members and customers to assess and review the project’s progress throughout its development.

Planning

- The first phase of Extreme Programming life cycle is planning, where customers or users meet with the development team to create 'user stories' or requirements.
- The development team converts user stories into iterations that cover a small part of the functionality or features required.
- A combination of iterations provides the customer with the final fully functional product.
- The programming team prepares the plan, time, and costs of carrying out the iterations, and individual developers sign up for iterations.
- One planning approach is the critical path method, grouping iterations essential for project progress in a linear fashion, and arranging for completion of other iterations parallel to the critical path.

Testing

- Extreme program integrates testing with the development phase rather than at the end of the development phase.
- All codes have unit tests to eliminate bugs, and the code passes all such unit tests before release.
- Another key test is customer acceptance tests, based on the customer specifications.
- Acceptance test run at the completion of the coding, and the developers provide the customer with the results of the acceptance tests along with demonstrations.

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 behaviour to remove duplication, improve communication, simplify, or add flexibility. This is called Refactoring.
- Code tends to rot. As we add feature after feature and deal with bug after bug, the structure of the code degrades. Left unchecked, this degradation leads to a tangled, unmaintainable mess.
- XP teams reverse this degradation through frequent refactoring.
- Refactoring is the practice of making a series of tiny transformations that improve the structure of the system without affecting its behavior.

Continue..

- After each tiny transformation, we run the unit tests to make sure that we haven't broken anything.
- Then we do the next transformation, and the next, and the next, running the tests after each.
- In this manner, we keep the system working while transforming its design.
- Refactoring is done continuously rather than at the end of the project, the end of the release, or the end of the iteration, or even the end of the day.
- Refactoring is something we do every hour or every half hour. Through refactoring, we continuously keep the code as clean, simple, and expressive as it can be.
- Prompts the developers to proactively improve the product as a whole
- Increases the developer knowledge of the system