ASSIGNMENT

EMERGING PROGRAMMING PARADIGMS

NIPUN HARJAI

2017UCO1694

COE - III

Scrum

Scrum is an agile process that is used for developing and maintaining complex projects, used primarily in the development of software.

It consists of multiple development iterations of fixed length, called sprints. These sprints generally last for one to two weeks each. The aim of these fixed length sprints is to ensure some feature is shipped at the end of each cycle. The team can then decide what features are to be shipped in the next sprint.

Scrum consists of a fixed set of responsibilities, meetings and roles that do not change. The roles of scrum include:

1. Product Owner
2. Scrum Team
3. Scrum Master

The product owner is responsible for having a view of the entire project and conveying the same to the team members. The product owner focuses on business as well as market requirements and lays down a priority for the work that needs to be done. Handling the backlog and providing guidance on what feature to ship next are also responsibilities of the product owner.

The scrum team comprises of five to ten members. Each of the members of the scrum team work together without any designated roles like tester and developer. They plan together for what features to ship in the next sprint by estimating what work they can complete in the next iteration.

The scrum master can often be considered the coach of the team and helps the team do their best possible work together. They work

closely with the product owner to ensure the product backlog is ready for the next sprint. Other responsibilities of the scrum master are organising meetings and dealing with issues the team might've encountered.

Some of the advantages of a scrum based approach include:

1. Increased accountability of the team, as the team themselves decide what work they will be completing in the next sprint, thus reducing the possibilities of micro-management from the project manager.
2. Constant communication between the team members ensure that all team members are aware of all existing issues and changes made.
3. Developing and testing features in small modules makes it easier to receive constant feedback and reduces the cost of fixing mistakes.
4. Daily standup meetings ensure the entire team knows what each team member is doing, thus reducing the possibility of confusion. Problems may be identified in earlier stages of development and are thus easier to resolve.

Lean Development

The concept of lean development was initially introduced by Toyota to bring reductions in wastage in their manufacturing process but was soon adopted by IT companies for their development processes. Lean development, in short, implies removing all unnecessary things from the development cycle.

Lean development is made possible by applying the following seven principles:

1. Elimination of waste: This is the most important principle in lean development. Elimination of waste includes improvements like removal of unnecessary code, unclear requirements, tasks in the backlog that cannot be completed in time, issues with quality, duplication of data, and features that aren't required.
2. Efficient management of quality: Repeated code testing, issues in logging and resolving these issues can significantly increase the cost of development and thus must be addressed. These issues may be tackled by using pair programming (completing each task in pairs) and test driven development (writing unit tests before code).
3. Amplification of knowledge: The amplification of knowledge is carried out through code reviews, establishing metrics from data that are cross-team and meetings.
4. Delaying commitment: This implies that all irreversible decisions are delayed as much as possible to leave room for improvement, as there is usually a change in requirements further down the development cycle.
5. Delivering fast: The concept of MVP, or, Minimum Viable Product, is used to ensure all unnecessary features are kept out of the development cycle, thus reducing complexity and enhancing delivery speed of critical features.
6. Respecting Team: Lean development aims to empower team members, instead of micromanaging them. This helps improve motivation and trust among members.
7. Optimising complete stream: Lean development aims at optimisation of entire systems rather than a part by part optimisation of the system by different teams. This results in a higher understanding of the entire system and it's working and thus higher throughput.

Extreme Programming

Extreme programming (XP) is a development method created to improve the quality of the software developed and its ability to adapt to any requirement changes down the development cycle. Just like Agile, extreme programming aims to provide iterative and frequent small releases, allowing both the team members and clients to review the project's progress throughout the development lifecycle.

Extreme Values

These are 5 fundamental values that define how the entire extreme programming paradigm is created. The 5 values are:

1. Simplicity
2. Communication
3. Feedback
4. Respect
5. Courage

Extreme Rules

The set of rules for extreme programming were designed to counter the claims that extreme programming will not be a viable option in modern software development. These rules are summarised below:

1. Planning (Writing user stoties, making frequent releases, dividing the project into iterations, etc.)
2. Managing (Setting a sustainable pace, measuring project velocity, giving the team a good work space, etc.)
3. Designing (Simplicity, not adding any functionality early, refactoring whenever possible, etc.)

4. Coding (Pair programming, integrating often, collective ownership, test driven development, etc.)
5. Testing (All modules must have unit tests, all modules must pass their unit tests before they are pushed into production, more tests are created when a bug is found, etc.)

Extreme Practices

Created using what were considered the best practices of software development at the time, these twelve Extreme Programming Best Practices detail the specific procedures that should be followed when implementing a project using Extreme Programming:

1. Pair programming
2. Planning game
3. Test driven development
4. Whole team
5. Continuous integration
6. Code refactoring
7. Small releases
8. Coding standards
9. Collective code ownership
10. Simple Design
11. System metaphor
12. Sustainable pace

Adaptive Software Development

Adaptive software development refers to the software development methodology of adding new features iteratively and incrementally to a complex project, while heavily factoring in the feedback received. The feedback is generally obtained from the client. In

other words, this approach aims to improve the solution based on continuous input received from the client.

This is a preferred approach over traditional development models as it leaves scope for improvement and does not focus on creating a solution in one go and delivering it to the client. Modern business environments are highly dynamic in nature and there is a need to maintain some wiggle room to incorporate all of the changing requirements. This allows the software solution to be modified according to the changing needs of the client.

There are 3 stages in adaptive software development lifecycle:

1. Speculate
2. Collaborate
3. Learn

Speculate is the project planning phase where the development team thoroughly understand the set of requirements of the client. This phase covers all of the tasks that need to be performed to complete the project, along with determining the deadline for each.

Collaborate is the stage where the development team works together to accomplish the tasks determined in the speculation phase. This phase also includes continuous communication with the client.

Learn is the stage where a prototype of the software with the newly added feature is deployed and given to the client for testing. The client then shares their feedback with the development team for further speculation and improvement.

Feature Driven Development

Feature Driven Development, or FDD, is an approach that promotes software development with close involvement of the product owner and all of the stakeholders of the project.
FDD was designed from scratch while keeping in mind larger teams, as working with larger teams poses its own challenges that must be tackled for efficient throughput.

FDD consists of five processes, the first three of which are equivalent to iteration zero, or, initial project-wide activities that help the development team put into place whatever they need to start delivering client-valued functionality in subsequent iterations.

The five processes in feature driven development are:

1. Developing an overall model
2. Building a features list
3. Planning by feature
4. Designing by feature
5. Building by feature

Building an initial object model in feature driven development is a highly collaborative and iterative approach. The main idea behind the same is for both the development and domain members of the development team gain a good understanding of the problem they are trying to solve.

After a model is developed, a feature list must be defined. A feature can be denoted by the form: <action> <result> <object>. A feature typically takes 1-3 days to implement, occasionally extending to 5 days.

The third FDD process, planning by feature, involves making an initial schedule and assigning responsibilities in the team. The feature list is given an order (or sequence) in which the features will be incorporated. This sequence may be adjusted later on in the development life cycle based on the input obtained from the client.

FDD focuses on individual code ownership rather than the collective code ownership that extreme programming aims at. This is because it may become harder to truly collectively maintain code ownership in large teams working on large projects. The pros of feature driven development are that it is driven from a functionality perspective. Its main purpose is to deliver software repeatedly in a timely manner by creating a prioritised feature list and plan development based on the same.