EPP ASSIGNMENT
NAME: ANKUSH GROVER
ROLL NO: 2017UCO1675

Write short notes on following

· Scrum

Scrum is an agile project management methodology or framework used primarily for software development projects with the goal of delivering new software capability every 2-4 weeks. It is one of the approaches that influenced the Agile Manifesto, which articulates a set of values and principles to guide decisions on how to develop higher-quality software faster.

Scrum addresses complexity in work by making information transparent, so that people can inspect and adapt based on current conditions, rather than predicted conditions. This allows teams to address the common pitfalls of a waterfall development process: chaos resulting from constantly changing requirements; underestimation of time, resources and cost; compromises on software quality; and inaccurate progress reporting. Transparency of common terms and standards is required in Scrum development to ensure that what is being delivered is what was expected. Frequent inspection ensures progress and detects variances early on so that adjustments can be made quickly.

Components of Scrum Development

The Scrum Team

- The Product Owner

The Product Owner is the project's key stakeholder or a spokesperson for the customer. There is only one Product Owner who conveys the overall mission and vision of the product which the team is building.

- The ScrumMaster

With no hierarchial authority over the team but rather more of a facilitator, the ScrumMaster ensures that the team adheres to Scrum theory, practices, and rules. The ScrumMaster protects the team by doing anything possible to help the team perform at the highest level. This may include removing impediments, facilitating meetings, and helping the Product Owner groom the backlog

- The Development Team

The Development Team is a self-organizing, cross-functional group armed with all of the skills to deliver shippable increments at the completion of each sprint. There are no titles in the Development Team and no one, including the ScrumMaster, tells the Development Team how to turn product backlog items into potentially shippable increments.

Scrum Events (Ceremonies)

- The Sprint

A sprint is a time-boxed period during which specific work is completed and made ready for review. Sprints are usually 2-4 weeks long but can be as short as one week.

- Sprint Planning Sprint

Planning team meetings are time-boxed events that determine which product backlog items will be delivered and how the work will be achieved.

- The Daily Stand-up

The Daily Stand-up is a short communication meeting (no more than 15 minutes) in which each team member quickly and transparently covers progress since the last stand-up, planned work before the next meeting, and any impediments that may be blocking his or her progress.

- The Sprint Review

The Sprint Review is the "show-and-tell" or demonstration event for the team to present the work completed during the sprint. The Product Owner checks the work against pre-defined acceptance criteria and either accepts or rejects the work. The stakeholders or clients give feedback to ensure that the delivered increment met the business need.

- The Retrospective

The Retrospective, or Retro, is the final team meeting in the Sprint to determine what went well, what didn't go well, and how the team can improve in the next Sprint. Attended by the team and the ScrumMaster, the Retrospective is an important opportunity for the team to focus on its overall performance and identify strategies for continuous improvement on its processes.

Scrum Artifacts

Product Backlog

The product backlog is the single most important document that outlines every requirement for a system, project or product. The product backlog can be thought of as a to-do list consisting of work items, each of which produces a deliverable with business value. Backlog items are ordered in terms of business value by the Product Owner.

Sprint Backlog

A sprint backlog is the specific list of items taken from the product backlog which are to be completed in a sprint.

Increment

An Increment is the sum of all product backlog items that have been completed since the last software release. While it is up to the Product Owner to decide on when an increment is released, it is the team's responsibility to make sure everything that is included in an increment is ready to be released. This is also referred to as the Potentially Shippable Increment (PSI).

Scrum Rules

The rules of agile Scrum should be completely up to the team and governed by what works best for their processes. The best agile coaches will tell teams to start with the basic scrum events listed above and then inspect and adapt based on your team's unique needs so there is continuous improvement in the way teams work together.

· Lean Development

Lean Software Development (LSD) is an agile framework based on optimizing development time and resources, eliminating waste, and ultimately delivering only what the product needs. The Lean approach is also often referred to as the Minimum Viable Product (MVP) strategy, in which a team releases a bare-minimum version of its product to the market, learns from users what they like, don’t like and want to be added, and then iterates based on this feedback.

Many organizations have found the LSD methodology to be an excellent approach to software development because of its streamlining of the process and forcing the team to ruthlessly cut away any activity that doesn’t directly affect the final product. But an organization must have an outstanding development team, and trust that team implicitly, for this approach to be successful.

Additionally, by waiting until the last minute to make decisions, the cost of change remains much less. Iterative development is utilized to deliver new applications or enhancements as quick as possible. Integrity is built into the software to ensure architecture and that system components flow well together. Organizations incorporate lean development principles to achieve continuous improvement as changes are rapidly implemented.

Lean software development is a set of principles that can be applied to software development to decrease programming effort, budgeting, and defect rates by one third. The principles were adapted from lean manufacturing by Mary and Tom Poppendieck. This approach is beneficial to an organization because agile iterations eliminate extensive pre-planned specifications. User stories rather than large upfront specs are easily understood by each team member and simpler to communicate.

Lean development makes it possible to gain information straight from the source, therefore eliminating the common problem of producing software that does not address the customers' needs. Short iterations provide an opportunity to communicate small sets of plans up front and allow the team to make decisions in order to adapt to unforeseen circumstances. Organizations that have the ability to complete fast, simple improvements in the shortest time frame gain powerful decision-making benefits.

· Extreme programming (XP)

Extreme programming (XP) is one of the most important software development framework of Agile models. It is used to improve software quality and responsive to customer requirements. The extreme programming model recommends taking the best practices that have worked well in the past in program development projects to extreme levels.

Basic principles of Extreme programming: XP is based on the frequent iteration through which the developers implement User Stories. User stories are simple and informal statements of the customer about the functionalities needed. A User story is a conventional description by the user about a feature of the required system. It does not mention finer details such as the different scenarios that can occur. The development team may decide to build a Spike for some feature. A Spike is a very simple program that is constructed to explore the suitability of a solution being proposed. It can be considered similar to a prototype. Some of the basic activities that are followed during software development by using XP model are given below:

- **Coding:** The concept of coding which is used in XP model is slightly different from traditional coding. Here, coding activity includes drawing diagrams (modeling) that will be transformed into code, scripting a web-based system and choosing among several alternative solutions.
- **Testing:** XP model gives high importance on testing and considers it be the primary factor to develop a fault-free software.
- **Listening:** The developers needs to carefully listen to the customers if they have to develop a good quality software. Sometimes programmers may not have the depth knowledge of the system to be developed. So, it is desirable for the programmers to understand properly the functionality of the system and they have to listen to the customers.
- **Designing:** Without a proper design, a system implementation becomes too complex and very difficult to understand the solution, thus it makes maintenance expensive. A good design results elimination of complex dependencies within a system. So, effective use of suitable design is emphasized.
- **Feedback:** One of the most important aspects of the XP model is to gain feedback to understand the exact customer needs. Frequent contact with the customer makes the development effective.

- **Simplicity:** The main principle of the XP model is to develop a simple system that will work efficiently in present time, rather than trying to build something that would take time and it may never be used. It focuses on some specific features that are immediately needed, rather than engaging time and effort on speculations of future requirements.

· Adaptive Software Development (ASD)

Adaptive Software Development is a method to build complex software and system. ASD focuses on human collaboration and self-organisation.

Adaptive Software Development practices provide ability to accommodate change and are adaptable in turbulent environments with products evolving with little planning and learning.

Phases of ASD Life Cycle

Adaptive Software Development is cyclical like the Evolutionary model, with the phase names reflecting the unpredictability in the complex systems. The phases in the Adaptive development life cycle are −

- Speculate
- Collaborate
- Learn

These three phases reflect the dynamic nature of Adaptive Software Development. The Adaptive Development explicitly replaces Determinism with Emergence. It goes beyond a mere change in lifecycle to a deeper change in management style. Adaptive Software Development has a dynamic Speculate-Collaborate-Learn Lifecycle.

The Adaptive Software Development Lifecycle focuses on results, not tasks, and the results are identified as application features.

Speculate

The term plan is too deterministic and indicates a reasonably high degree of certainty about the desired result. The implicit and explicit goal of conformance to plan, restricts the manager's ability to steer the project in innovative directions.

In Adaptive Software Development, the term plan is replaced by the term speculate. While speculating, the team does not abandon planning, but it acknowledges the reality of uncertainty in complex problems. Speculate encourages exploration and experimentation. Iterations with short cycles are encouraged.

Collaborate

Complex applications are not built, they evolve. Complex applications require that a large volume of information be collected, analyzed, and applied to the problem. Turbulent environments have high rates of information flow. Hence, complex applications require that a large volume of information be collected, analyzed, and applied to the problem. This results in diverse Knowledge requirements that can only be handled by team collaboration.

Collaborate would require the ability to work jointly to produce results, share knowledge or make decisions.

In the context of project management, Collaboration portrays a balance between managing with traditional management techniques and creating and maintaining the collaborative environment needed for emergence.

Learn

The Learn part of the Lifecycle is vital for the success of the project. Team has to enhance their knowledge constantly, using practices such as −

- Technical Reviews
- Project Retrospectives
- Customer Focus Groups

Reviews should be done after each iteration. Both, the developers and customers examine their assumptions and use the results of each development cycle to learn the direction of the next.

We observe the following from an Adaptive framework.

- It is difficult to Collaborate without Learning or to Learn without Collaborating.
- It is difficult to Speculate without Learning or to Learn without Speculating.
- It is difficult to Speculate without Collaborating or to Collaborate without Speculating.

· Feature Driven Development

Feature Driven Development (FDD) is an agile framework that, as its name suggests, organizes software development around making progress on features. Features in the FDD context, though, are not necessarily product features in the commonly understood sense. They are, rather, more akin to user stories in Scrum. In other words, “complete the login process” might be considered a feature in the Feature Driven Development (FDD) methodology.

If you’re with a big corporation or are working on a large-scale software project, FDD might be right for your project. But this methodology relies heavily on chief developers and has a top-down decision-making approach, as opposed to some of the other agile frameworks that are based more on collective project ownership. If that type of methodology fits your company’s culture, then Feature Driven Development is worth investigating.

FDD was designed to follow a five-step development process, built largely around discrete “feature” projects. That project lifecycle looks like this:

1. Develop an overall model
2. Build a features list
3. Plan by feature
4. Design by feature
5. Build by feature

Pros of FDD:

- Simple five-step process allows for more rapid development
- Allows larger teams to move products forward with continuous success
- Leverages pre-defined development standards, so teams are able to move quickly

Cons of FDD:

- Does not work efficiently for smaller projects
- Less written documentation, which can lead to confusion
- Highly dependent on lead developers or programmers