Write short notes on following

Scrum

· Lean Development

· Extreme programming (XP)

· Adaptive Software Development (ASD)

· Feature Driven Development

1. **SCRUM**

Scrum includes the same types of activities as the waterfall methodology, but rather than implementing the Scrum is based on the principles of transparency to all stakeholders along with continuous inspection and adaptation to changing conditions. Scrum builds the application incrementally, with each increment adding and improving features and functionality created by its predecessors.

# Pieces & Parts

Using Scrum requires people specific Scrum-roles, information describing the features and functions of the application, information needed to ensure that the workings of the project is transparent to everyone on the team, and processes that govern the operation of the project team.

# People & Roles

There are three types of roles internal to a Scrum team and one that’s external. Two of the internal roles are:

Product Owner

The Product Owner is the primary interface between the team and the Business Owner, and is responsible for maintaining and prioritizing the backlog of tasks to be performed. This is performed transparently and collaboratively with the Development team.

Scrum Master

The primary goal of the Scrum Master is keep the Development team productive. From the perspective of the Product Owner the Scrum Master represents the Development team ensures that their issues, concerns, and roadblocks are taken into consideration by the Product Owner.

1. **LEAN DEVELOPMENT**

Lean development is the application of Lean principles to software development. Lean principles got their start in manufacturing, as a way to optimize the production line to minimize waste and maximize value to the customer. Of course, there are some major differences between manufacturing and software development, as well; namely, that manufacturing deals with the production of physical goods, while the value being created in software development is created within the mind of the developer.

There are 7 Lean Development Principles

The seven Lean principles are:

* Eliminate waste
* Build quality in
* Create knowledge
* Defer commitment
* Deliver fast
* Respect people
* Optimize the whole

### Eliminate waste

One of the key elements of practicing Lean is to eliminate anything that does not add value to the customer. There are seven wastes (or muda) defined in Each of these wastes should be systematically eliminated in order to maximize customer value:

* Unnecessary code or functionality: Delays time to customer, slows down feedback loops
* Starting more than can be completed: Adds unnecessary complexity to the system, results in context-switching, handoff delays, and other impediments to flow
* Delay in the software development process: Delays time to customer, slows down feedback loops

### Build quality in

In trying to ensure quality, many teams actually create waste – through excessive testing, for example, or an excessive logging of defects.

In Lean development, quality is everyone’s job, not just that of the quality analyst.

These are some of the most popular Lean development tools for building quality in:

* Pair programming: Avoid quality issues by combining the skills and experience of two developers instead of one
* Test-driven development: Writing criteria for code before writing the code to ensure it meets business requirements
* Incremental development and constant feedback
* Minimize wait states: Reduce context switching, knowledge gaps, and lack of focus

### Create knowledge

The Lean development principle of Create Knowledge is another one that seems simple, but requires discipline and focus to implement. This principle encourages Lean teams to provide the infrastructure to properly document and retain valuable learning. This can be done by using any combination of the following tools:

* Pair programming
* Code reviews
* Documentation
* Wiki – to let the knowledge base build up incrementally

Defer commitment

This Lean development principle is easily misused. This Lean principle encourages team to demonstrate responsibility by keeping their options open and continuously collecting information, rather than making decisions without the necessary data.

To defer commitment means to:

* Not plan (in excessive detail) for months in advance
* Not commit to ideas or projects without a full understanding of the business requirements
* Constantly be collecting and analyzing information regarding any important decisions

### Deliver fast

Every team wants to deliver fast, to put value into the hands of the customer as quickly as possible. Here are a few common problems :

* Thinking too far in advance about future requirements
* Blockers that aren’t responded to with urgency
* Over-engineering solutions and business requirements

Lean development is based on this concept: Build a simple solution, put it in front of customers, enhance incrementally based on customer feedback. This is important, especially in software, because speed to market is an incredible competitive advantage.

1. **Extreme Programming (XP)**

Definition

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.

When Applicable

The general characteristics:

* Dynamically changing software requirements
* Risks caused by fixed time projects using new technology
* Small, co-located extended development team
* The technology you are using allows for automated unit and functional tests

Values

The five values of XP are communication, simplicity, feedback, courage, and respect and are described in more detail below.

Communication

Software development is inherently a team sport that relies on communication to transfer knowledge from one team member to everyone else on the team.

Simplicity

Simplicity means “what is the simplest thing that will work?” The purpose of this is to avoid waste and do only absolutely necessary things such as keep the design of the system as simple as possible so that it is easier to maintain, support, and revise.

Feedback

Through constant feedback about their previous efforts, teams can identify areas for improvement and revise their practices. Feedback also supports simple design.

Courage

Kent Beck defined courage as “effective action in the face of fear” (Extreme Programming Explained P. 20). This definition shows a preference for action based on other principles so that the results aren’t harmful to the team.

Respect

The members of your team need to respect each other in order to communicate with each other, provide and accept feedback that honors your relationship, and to work together to identify simple designs and solutions.

Practices

The core of XP is the interconnected set of software development practices listed below. While it is possible to do these practices in isolation, many teams have found some practices reinforce the others and should be done in conjunction to fully eliminate the risks you often face in software development.

The XP Practices have changed a bit since they were initially introduced.The original twelve practices are listed below.

* Small Releases
* The Planning Games
* Metaphor
* Simple Design
* Testing
* Refactoring
* Pair Programming
* Collective Ownership
* Continuous Integration
* 40-hour week
* On-site Customer
* Coding Standard

1. **ADAPTIVE SOFTWARE DEVELOPMENT**

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.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. 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.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

1. **Feature-driven development (FDD)**

Feature-driven development (FDD) is a customer-centric software development methodology known for short iterations and frequent releases. Like Scrum, FDD requires the customer, also known as the project business owner, to attend the initial design meeting and iteration retrospective.

Steps of feature-driven development

Feature-driven development is made up of five basic process steps:

1. Develop a model- The overall model is created by the chief architect, or another professional leading the project, by identifying the scope and context of the system.
2. Build a list of features- Developers brainstorm a list of potential items that would be useful to users and could be completed along a set timeline for release. Each feature should be manageable within a timeframe of around two weeks.
3. Plan out each feature- Features are organized by how long they take to create and how important they are to the client. Ownership of each feature is also assigned.
4. Design each feature- The actual details of each feature are produced, inspected and finalized.
5. Build each feature- After the design is improved, the completed feature is added to the official build for delivery to the client.

Best practices for feature-driven development

To achieve the highest level of success, FDD is built around overall software engineering best practices. This includes:

* Identifying the domain object model, or the scope of the problem that needs to be solved, to help with the framework for feature development.
* Breaking down complex features into smaller functions and subsets.
* Assigning features to a single owner to ensure consistency and code integrity.
* Building dynamic and diverse feature teams to collect multiple design options.
* Performing routine code inspections of each feature before implementation into the main build.